Semantic Data Management with Applications in data-driven Chemistry

Von der Fakultät für Informatik
der RWTH Aachen University zur Erlangung
des akademischen Grades eines

Doktors der Naturwissenschaften
genehmigte Dissertation vorgelegt von

Sayed Massih Hoseini
aus Neuss, Deutschland

Berichter: Professor Dr. rer-pol. Stefan Decker
Professor Dr. rer. nat. Christoph Quix

Tag der mündlichen Prüfung: 16. September 2026

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Abstract

Scientific and industrial environments increasingly span experimental, analytical, and production-oriented processes, bringing together data that differs in format, structure, provenance, and level of abstraction. Conventional data architectures provide important foundations for scalable storage and processing, but they do not resolve the semantic heterogeneity that limits data access, integration, and interpretability. This dissertation argues that treating semantics as a core architectural concern is the key to unlocking the analytical potential of heterogeneous data.

This argument is developed in four connected stages. First, a conceptual framework for characterizing data architectures is established and their suitability for heterogeneous industrial environments is evaluated. Second, the Semantic Data Reservoir (SEDAR) is introduced as a semantic data platform integrating semantic data access and automated analytics orchestration. Third, a formal framework for annotating, relating, and querying heterogeneous data sources in terms of domain knowledge is developed. Fourth, the potential of automated machine learning and large language model interaction to lower the barrier for non-technical users performing complex data-management and analytics tasks is demonstrated.

The proposed concepts are realized in SEDAR and evaluated through application scenarios from chemistry. The results show that ontology-based semantics and workflow automation can improve the usability of data-driven research. They further show that semantic infrastructure, combined with automated machine learning, enables competitive performance on domain-specific tasks without extensive manual model engineering.

Kurzfassung

Wissenschaftliche und industrielle Umgebungen umfassen zunehmend experimentelle, analytische und produktionsorientierte Prozesse und führen damit Daten zusammen, die sich in Format, Struktur und Abstraktionsgrad unterscheiden. Konventionelle Datenarchitekturen bilden eine wichtige Grundlage für skalierbare Speicherung und Verarbeitung, lösen jedoch nicht die semantische Heterogenität, die den Datenzugang, die Integration und die Interpretierbarkeit einschränkt. Diese Dissertation vertritt die Auffassung, dass die Behandlung von Semantik als zentrales architektonisches Anliegen der Schlüssel dazu ist, das analytische Potenzial heterogener Daten zu erschließen.

Dieses Argument wird in vier miteinander verbundenen Schritten entwickelt. Erstens wird ein konzeptioneller Rahmen zur Charakterisierung von Datenarchitekturen etabliert und ihre Eignung für heterogene industrielle Umgebungen bewertet. Zweitens wird das Semantic Data Reservoir (SEDAR) als semantische Datenplattform eingeführt, die semantischen Datenzugriff und die automatisierte Orchestrierung analytischer Prozesse integriert. Drittens wird ein formaler Rahmen für die Annotation, Verknüpfung und Abfrage heterogener Datenquellen in Begriffen des Domänenwissens entwickelt. Viertens wird das Potenzial des automatisierten maschinellen Lernens und der Interaktion mit großen Sprachmodellen aufgezeigt, die Hürde für nichttechnische Nutzer bei der Durchführung komplexer Datenmanagement- und Analyseaufgaben zu senken.

Die vorgeschlagenen Konzepte werden in SEDAR umgesetzt und anhand von Anwendungsszenarien aus der Chemie evaluiert. Die Ergebnisse zeigen, dass ontologiebasierte Semantik und Workflow-Automatisierung die Nutzbarkeit datengetriebener Forschung verbessern können. Sie zeigen ferner, dass semantische Infrastruktur in Verbindung mit automatisiertem maschinellem Lernen eine hohe Leistungsfähigkeit bei domänenspezifischen Aufgaben ermöglicht, ohne umfangreiche manuelle Modellentwicklung zu erfordern.

Acknowledgements

This dissertation was written during my time as a research associate at the Niederrhein University of Applied Sciences in Krefeld, Germany. My PhD is formally affiliated with the Chair of Databases and Information Systems (i5) at the RWTH Aachen University. During this period, many people supported me in my work on this thesis, and I would like to take this opportunity to express my sincere gratitude.

- I would like to thank Prof. Christoph Quix for his intensive supervision, his many valuable suggestions and comments, and for providing the opportunity to work on a research project in an inspiring environment.
- My sincere thanks also go to Prof. Stefan Decker for kindly agreeing to act as first supervisor and for his valuable feedback, particularly during the final phase of this dissertation.
- I am also grateful to my colleagues at the university, in particular Jens Kaufmann, Gaoyuan Zhang, Dominik Polke, Alvin Surjana, and Christian Schmitz, for numerous discussions and valuable input on both this work and many other topics.
- My thanks also go to my thesis students and student assistants, especially Vincent Herrmann, Max Knoll, Max Ibbels, Alexander Martin, Leon Beer, Marcel Thiel, Benjamin Danker, and Nico Kuth, for their support in developing and implementing parts of this work.
- I would also like to thank Elske Schönhals for the excellent collaboration within the PEDaL HN research project.
- Furthermore, I thank all project partners involved in the i²DACH research project, which was funded by the German Federal Ministry of Education and Research within the program “Forschung an Fachhochschulen” (grant no. 13FH557KX0).

Finally, I would like to express my deepest gratitude to my parents, who made my studies possible and have always supported me. I also thank my wife and my siblings, from whom I have learned a great deal and who have always stood by me.

Hamburg, September 2026 Sayed Massih Hoseini

Declaration of Authorship

I, Sayed Massih Hoseini
declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.
I do solemnly swear that:

- 1. This work was done wholly or mainly while in candidature for the doctoral degree at this faculty and university;
- 2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
- 3. Where I have consulted the published work of others or myself, this is always clearly attributed;
- 4. Where I have quoted from the work of others or myself, the source is always given. This thesis is entirely my own work, with the exception of such quotations;
- 5. I have acknowledged all major sources of assistance;
- 6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- 7. Parts of this work have been published before. A detailed list can be found on the following page.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, I used generative artificial intelligence to improve the writing, make suggestions, and for rephrasing. After using these tools & services, I carefully reviewed and edited the content as needed. I take full responsibility for the content of this thesis.

Remarks with regards to reading this document digitally

This thesis is intended to be read as a digital-first document. As such, it makes extensive use of hyperlinks throughout the text and employs color schemes and figures optimized for on-screen reading. In most PDF viewers, e.g. Acrobat Reader by Adobe, on your keyboard you can use `Alt` + ← to return after following a hyperlink.

Publications

The following publications constitute the main scientific basis of this dissertation. They are not reproduced as a cumulative dissertation, but individual results, concepts, systems, and evaluation scenarios developed in these publications are incorporated and extended in the corresponding chapters. In all publications listed in this subsection, I contributed substantially to the conception, implementation, evaluation, and writing; publications in which I am first author represent work primarily led by me.

Publications forming the basis of the dissertation

[Hoseini et al., 2024e] Sayed Hoseini, Gaoyuan Zhang, Dominik Polke, Alvin Surjana, Lasse Wagner, Christian Schmitz, and Christoph Quix. Coatings Intelligence: Data-driven Automation for Chemistry 4.0. IEEE 7th International Conference on Industrial Cyber-Physical Systems, 2024.

[Zhang et al., 2022] Gaoyuan Zhang, Christian Schmitz, Matthias Fimmers, Christoph Quix, and **Sayed Hoseini**. Deep learning-based automated characterization of crosscut tests for coatings via image segmentation. Journal of Coatings Technology and Research, 2022.

[Hoseini et al., 2022] Sayed Hoseini, Gaoyuan Zhang, Alexander Jongbloed, Christian Schmitz, and Christoph Quix. Automated defect detection for coatings via height profiles obtained by laser-scanning microscopy. Machine Learning with Applications, 2022.

[Hoseini et al., 2024c] Sayed Hoseini, Maximilian Ibbels, and Christoph Quix. Enhancing Machine Learning Capabilities in Data Lakes with AutoML and LLMs. European Conference on Advances in Databases and Information Systems, 2024.

[Hoseini et al., 2024d] Sayed Hoseini, Johannes Theissen-Lipp, and Christoph Quix. A survey on semantic data management as intersection of ontology-based data access, semantic modeling and data lakes. Journal of Web Semantics, 2024.

[Hoseini et al., 2023] Sayed Hoseini, Ahmed Ali, Haron Shaker, and Christoph Quix. SEDAR: A Semantic Data Reservoir for Heterogeneous Datasets. Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023.

[Hoseini et al., 2024b] Sayed Hoseini, Andreas Burgdorf, Alexander Paulus, Tobias Meisen, Christoph Quix, and André Pomp. Towards LLM-augmented Creation of Semantic Models for Dataspaces. The Second International Workshop on Semantics in Dataspaces, co-located with the Extended Semantic Web Conference, 2024.

[Hoseini et al., 2024a] Sayed Hoseini, Andreas Burgdorf, Alexander Paulus, Tobias Meisen, Christoph Quix, and André Pomp. Challenges and Opportunities of LLM-Augmented Semantic Model Creation for Dataspaces. In European Semantic Web Conference, pages 183–200. Springer, 2024.

[Hoseini, 2024] Sayed Hoseini. Semantic Data Management for managing heterogeneous data sources in Chemistry 4.0. Doctoral Consortium at ISWC 2024 co-located with the

23rd International Semantic Web Conference, 2024.

[Hoseini et al., 2026a] **Sayed Hoseini**, Christoph Quix, and Stefan Decker. Conversational data management: A multi-agent natural language interface for semantic data lakes. European Conference on Advances in Databases and Information Systems, 2026.

[Hoseini et al., 2025a] **Sayed Hoseini**, Vincent Herrmann, and Christoph Quix. End-to-end ML with LLMs and semantic data management: Experiences from Chemistry 4.0. In Proceedings of the Workshop on Data Management for End-to-End Machine Learning (DEEM '25) in Berlin, Germany, 2025.

[Hoseini et al., 2025b] **Sayed Hoseini**, Maximilian Ibbels, Maximilian Knoll, and Christoph Quix. Towards Enhancing Data Science Agents with Semantics. Computer Science and Information Systems, 2025.

Secondary Publications

For completeness, the following list contains further publications that are related to my doctoral research environment but are not directly incorporated into the dissertation:

[Sharma et al., 2025] Anmol Sharma, Sulayman K. Sowe, Soo-Yon Kim, **Sayed Hoseini**, Fidan Limani, Zeyd Boukhers, Christoph Lange, and Stefan Decker. FAIR data assessment using LLMs: The Fair-Way. In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. ACM, 2025.

[Kaufmann et al., 2021a] Jens Kaufmann, **Sayed Hoseini**, Pascal Quindeau, Christoph Quix, and Sylvia Ruschin. Bringing it all together–gemeinschaftlich aktiv lernen am virtuell geteilten Bildschirm in der Hochschule und digital. HMD Praxis der Wirtschaftsinformatik, 2021.

[Quindeau et al., 2022] Pascal Quindeau, **Sayed Hoseini**, Jens Kaufmann, and Christoph Quix. Einsatz einer webbasierten Softwarelösung zur Echtzeitkollaboration in technisch-orientierten Studienfächern. Gesellschaft für Informatik e.V., 2022.

[Kaufmann et al., 2021b] Jens Kaufmann, Stephan Kühnel, René Theuerkauf, Emal M Alekozai, **Sayed Hoseini**, Uwe Neuhaus, and Michael Schulz. Where is the science in data science projects? Gesellschaft für Informatik, 2021.

[Schulz et al., 2022] Michael Schulz, Uwe Neuhaus, Jens Kaufmann, Stephan Kühnel, Emal M Alekozai, Heiko Rohde, **Sayed Hoseini**, René Theuerkauf, Daniel Badura, Ulrich Kerzel, et al. DASC-PM v1.1 – Ein Vorgehensmodell für Data-Science-Projekte. Universitäts- und Landesbibliothek Sachsen-Anhalt, 2022.

[Theuerkauf et al., 2022] René Theuerkauf, Stephan Daurer, **Sayed Hoseini**, Jens Kaufmann, Stephan Kühnel, Florian Schwade, Emal M Alekozai, Uwe Neuhaus, Heiko Rohde, and Michael Schulz. Vorschlag eines morphologischen Kastens zur Charakterisierung von Data-Science-Projekten. Informatik Spektrum, 2022.

[Schulz et al., 2023] Michael Schulz, Uwe Neuhaus, Stephan Kühnel, Heiko Rohde, **Sayed Hoseini**, and René Theuerkauf. DASC-PM v1.1 Case Studies. Universitäts- und Landesbibliothek Sachsen-Anhalt, 2023.

[Wicht et al., 2021] Patric Wicht, Alexandra Hausmann, **Sayed Hoseini**, and Jens Kaufmann. Aufbau eines Data-Science-Teams - "Lessons learned". Wirtschaftsinformatik & Management, 2021.

[Polke et al., 2024] Dominik Polke, Alvin Surjana, Gaoyuan Zhang, **Sayed Hoseini**, Lasse Wagner, Carmen Stoffelen-Janßen, Christian Schmitz, Jost Göttert, Christoph Quix, and Elmar Ahle. Entwicklung eines Automatisierungsframeworks zur Mehrgrößenoptimierung in chemischen Hochdurchsatz-Prozessen. VDI AUTOMATION, 2024.

[Schönhals and Hoseini, 2025] Elske M. Schönhals and **Sayed Hoseini**. Is the Data Awareness Rally Really Fun? Exploring Assumptions in Course Design by Analysing Student Feedback. In Niederrhein University (PEDaL HN project), 2025.

[Polke et al., 2026] Dominik Polke, Alvin Immanuel Surjana, **Sayed Hoseini**, Lasse Wagner, Jost Göttert, Christoph Quix, and Elmar Ahle. Development of an Automation Framework for Data-Driven Modeling and Adaptive Learning in Industrial Processes. 9th IEEE International Conference on Industrial Cyber-Physical Systems, 2026.

[Zhang et al., 2026] Gaoyuan Zhang, Christoph Quix, **Sayed Hoseini**, and Christian Schmitz. Natural-Language Interfaces to a Process-Aware, Graph-Based ELN: A UV Coatings Case Study. Digital Chemical Engineering, 2026.

Supervised Theses

Parts of this dissertation are based on the following bachelor and master theses that I supervised. For the theses where I was the first advisor or second supervisor, I conceived the project idea, goals, and basic architecture and evaluation design, while my co-advisors provided feedback and domain expertise. As a secondary advisor, I provided feedback and expertise in the same way. The student developed the detailed thesis concept, implementation and evaluation, and the thesis report. All advisors provided feedback and reviewed the solutions and manuscripts.

List of Supervised Theses

[Hildebrandt, 2021] Yannik Hildebrandt. Behavior Driven Development: Analysis of a Company. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2021. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Danker, 2021] Benjamin Danker. Detection of Defects on Paint Surfaces Using Machine Learning. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2021. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Martin, 2022] Alexander Martin. Design and Implementation of a Generic Ingestion Interface with Versioning for Data Lake Systems. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2022. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Wegmeyer, 2022] Jan-Philipp Wegmeyer. Development of a Concept for Data Science Projects in Smart Cities. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2022. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Burgsmüller, 2022] Frederik Burgsmüller. Development of a Software Library for Few-Shot Learning Based on the CrossTransformer Model. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2022. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Thiel, 2022] Marcel Thiel. Development of a Semantic Metadata Management System and its Integration into a Data Lake Prototype System. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2022. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Shaker, 2023] Haron Shaker. Generation of Semantic Models in a Semantic Data Lake using GNNs. Master's thesis, RWTH Aachen University, 2023. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Advisor: Sayed Hoseini, M.Sc.

[Ali, 2023] Ahmed Ali. Integration of an Advanced User Interface for Semantic Data Management into a Semantic Data Lake. Master's thesis, RWTH Aachen University, 2023. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Advisor: Sayed Hoseini, M.Sc.

[Kuth, 2023] Nico Kuth. Design and Implementation of a Programming Interface for the Data

Lake System SEDAR. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2023. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Beer, 2023] Leon Beer. Enhancement of a Data Lake System for MLOps. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2023. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Ibbels, 2024] Maximilian Ibbels. Extension of a Data Lake System with Automated Machine Learning. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2024. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Mittelstädt, 2025] Niels Mittelstädt. An Agent-Based Natural Language Interface for the Semantic Data Lake System SEDAR. Master's thesis, RWTH Aachen University, 2025. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Advisor: Sayed Hoseini, M.Sc.

[Knoll, 2025] Maximilian Knoll. Entwicklung eines LLM-basierten Data-Science-Agenten zur automatisierten Modellgenerierung. Master's thesis, Hochschule Niederrhein University of Applied Sciences, 2025. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Sharma, 2025] Anmol Sharma. LLM-based Tool for FAIR Data Assessment. Master's thesis, RWTH Aachen University, 2025. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Advisor: Sayed Hoseini, M.Sc.

[Herrmann, 2024] Vincent Herrmann. LLM-basierte Code-Generierung für Machine-Learning-Pipelines mit semantischen Beschreibungen. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2024. Supervisors: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

[Dubrov, 2025] Erik Dubrov. From Data Integration to Financial Insights: Analysing a Centralized Data Platform and Its Use in Power BI Reporting. Bachelor's thesis, Hochschule Niederrhein University of Applied Sciences, 2025. Written in collaboration with SWK AG. Examiners: Prof. Dr. Christoph Quix, Sayed Hoseini, M.Sc.

Contents

1 Introduction **1**
1.1 Research Environment and Motivation . . . 2
1.2 Research Questions, Contributions and Outline . . . 5

2 Theoretical Background **9**
2.1 Data . . . 10
2.2 Data Management . . . 13
2.3 Metadata management . . . 15
2.4 The FAIR Data Principles . . . 17
2.5 Data Governance . . . 17
2.6 Data Integration . . . 18
2.7 Data Quality Management . . . 20
2.7.1 Data Profiling . . . 20
2.7.2 Data Cleaning . . . 21
2.8 Data Analytics . . . 21
2.8.1 Large Language Models . . . 23
2.9 Semantic Data Management . . . 25

3 Defining Data Architectures and their Technical Requirements **37**
3.1 Introduction . . . 38
3.2 Terminology and Requirements for Data Architectures . . . 40
3.3 Data Architectures . . . 42
3.3.1 Data Warehouse . . . 44
3.3.2 Data Lake . . . 46
3.3.3 Semantic Data Lake . . . 49
3.3.4 Data Lakehouse . . . 51
3.3.5 Data Fabric . . . 54
3.3.6 Data Mesh . . . 58
3.4 Evaluation . . . 61
3.4.1 Dimensional Analysis . . . 62
3.4.2 Evaluation of Real-World Systems . . . 63
3.5 Related Surveys & Work . . . 65

3.6 Conclusion . . . 67

4 The Semantic Data Reservoir (SEDAR) 69
4.1 Motivation and Research Context . . . 70
4.2 Research Objectives . . . 71
4.3 Implementation . . . 72
4.3.1 Ingestion Layer . . . 73
4.3.2 Storage Layer . . . 79
4.3.3 Transformation Layer . . . 83
4.3.4 Interaction Layer . . . 85
4.4 Conclusion . . . 88

5 Managing Semantics in SEDAR 89
5.1 Introduction . . . 90
5.2 Ontology-based Data Management . . . 92
5.2.1 Semantic Labels . . . 93
5.2.2 Relation to RML (executable semantic labeling). . . . 100
5.2.3 Relation to SemTab (CTA/CEA/CPA). . . . 101
5.2.4 Semantic Models . . . 102
5.3 Semantic Type Detection with LLMs . . . 103
5.4 Using Knowledge Graphs with LLMs for Semantic Model Creation . . . 105
5.5 Semantic Modeling Platforms . . . 107
5.6 Ontology-based Data Access & Integration . . . 112
5.6.1 Query Rewriting vs. Materialization . . . 112
5.6.2 The OBDA Architecture: Ontology, Mappings, and Query Translation . . . 114
5.6.3 Historical Evolution of OBDA and Summary . . . 116
5.7 Implementation in SEDAR . . . 118
5.7.1 Ontology & KG Management . . . 118
5.7.2 Creating Semantic Labels with LLMs in SEDAR . . . 118
5.7.3 Semantic Modeling Workflow in SEDAR . . . 123
5.7.4 Ontology-based Data Access in SEDAR . . . 124
5.8 Conclusion . . . 127

6 Automating Machine Learning in SEDAR 129
6.1 Conceptual Foundations of Automated ML in SEDAR . . . 130
6.1.1 AutoML Abstraction . . . 131
6.1.2 Implementation . . . 133
6.2 Data Science Agents . . . 137
6.2.1 Related Work . . . 137
6.2.2 Neuro-Symbolic Data Science Agents . . . 138
6.2.3 Integrating Knowledge Graphs with Data Science Agents . . . 140
6.2.4 Implementation of a KG-enhanced Data Science Agent . . . 141

6.3 Conclusion . . . 145

7 A Multi-Agent Natural Language Interface for SEDAR 147
7.1 Introduction . . . 148
7.2 Related Work . . . 149
7.3 Extending the SEDAR Architecture . . . 150
7.4 Multi-Agent System: Concepts and Architecture . . . 151
7.5 Realization . . . 153
7.5.1 Multi-Agent System . . . 153
7.5.2 Retrieval Pipeline . . . 156
7.5.3 Chat Interface & Example Walkthrough . . . 158
7.5.4 Finetuning & Prompt Compression . . . 160
7.6 Evaluation & Discussion . . . 161
7.6.1 Quantitative evaluation . . . 161
7.6.2 Qualitative Evaluation. . . . 166
7.6.3 Impact of Retrieval Quality. . . . 168
7.7 Conclusion . . . 168

8 Applications of SEDAR in Data-Driven Chemistry 171
8.1 Introduction . . . 172
8.2 Object Detection on 3D Height Maps to Identify Coating Defects . . . 173
8.2.1 Sample Preparation . . . 174
8.2.2 Automated imaging . . . 174
8.2.3 Data Preprocessing . . . 176
8.3 Image Segmentation for Automating Characterization of Crosscut Tests for Coatings . . . 177
8.3.1 Sample Preparation . . . 177
8.3.2 Data Preprocessing . . . 178
8.4 Regression of Scratch Resistance from Sensor Data in a Smart Manufacturing Pipeline for Coatings . . . 179
8.4.1 Pipeline Automation . . . 180
8.4.2 Experimental Setup . . . 181
8.5 Evaluating SEDAR's automated machine learning capabilities . . . 185
8.5.1 Results of AutoMLWrapper and LLM-based Pipelines . . . 187
8.5.2 Results of the KG-enhanced Approach . . . 189
8.5.3 Discussion . . . 190
8.6 End-To-End ML with LLMs and Semantic Data Management . . . 190
8.6.1 Context-Aware Code Generation with LLMs and Ontology-Based Semantics . . . 194
8.6.2 Evaluation . . . 197
8.6.3 Discussion . . . 199

8.7 SEDAR's Research Contributions and Evaluation with DATER 200
8.8 Conclusion 202

9 Conclusion and Outlook **203**
9.1 Key contributions 203
9.2 Discussion, Limitations and Directions for Future Work 204

Appendix **253**

List of Figures

1.1 Proposed research in the i²DACH-project [Quix et al., 2023] 4
1.2 Various aspects in this work. 6

2.1 Data Management Activities as depicted by [Eichler et al., 2021] and based on the *DMBOK* [International, 2017]. 14
2.2 Tweaked Semantic Web Technology Layer Cake suggested by [Idehen, 2017], based on the original proposal of Tim Berners-Lee from 2000 [Berners-Lee, 2000]. 26

3.1 Overview of terminology, abstractions and components for data architectures. . . 41
3.2 Evolution of the various discussed data architectures 43
3.3 Data warehouse architecture based on [Jarke et al., 1999, 2003] 45
3.4 Four-layered data lake architecture [Jarke and Quix, 2017] 47
3.5 Semantic data lake architecture [Hoseini et al., 2024d] 47
3.6 Data Lakehouse architecture inspired by [Jarke and Quix, 2017, Armbrust et al., 2021] 53
3.7 Data fabric architecture inspired by [Priebe et al., 2021, Hechler et al., 2023] . . 57
3.8 Data mesh architecture inspired by [Eichler et al., 2022, Dolhopolov et al., 2024, Hechler et al., 2023, Strengholt, 2023] 60

4.1 Conceptual four-layered data lake architecture from section 3.3.2 73
4.2 SEDAR's architecture [Hoseini et al., 2023], as a concrete implementation of the four-layered data lake architecture. 73
4.3 *SEDAR*'s *Ingestion Layer* as depicted by Martin [2022]. 76
4.4 Data ingestion process. 77
4.5 Metadata extraction process 79
4.6 SEDAR's metadata model, colour-coded by architectural concern: semantic layer (green), data assets/representations (blue), lifecycle & provenance (red), governance context (purple), quality signals (yellow), and analytics/model lineage (orange). 81
4.7 Example of a data processing workflow defined in the *Workflow Module*. 84
4.8 Conceptual architecture of the Interaction Layer in SEDAR (left-to-right). 85

5.1 Phases to generate semantic models for OBDA [Paulus et al., 2021b, Hoseini et al., 2024d] . . . 90
5.2 A semantic model for heterogeneous data sources. Most concepts of the semantic model are projected (blue) from the KG (black), but to describe specific data sets one may define also novel elements (red). . . . 92
5.3 Screenshot of the modeling interface in PLASMA [University of Wuppertal, 2025] 109
5.4 Illustration of Ontology-Based Data Access [Hoseini et al., 2024d] . . . 114
5.5 Timeline of OBDA techniques [Hoseini et al., 2024d] . . . 117
5.6 Semantic Data Processing pipeline . . . 124
5.7 Generating one semantic model for multiple datasets . . . 124
5.8 Combining multiple datasets . . . 124
5.9 The *OBDA* tab displaying a SPARQL query to find currencies and capitals of countries. . . . 128

6.1 From abstract AutoML experiment specification to concrete execution in SEDAR. The configuration layer encodes an ontology-like schema reflecting the hierarchical structure of configuration files (data type, task type, problem type, and backend-specific classes) and associated parameter mappings. . . . 133
6.2 Excerpts of the global (left), the *AutoGluon*-specific configuration file (center) and exemplary code of the AutoMLWrapper for image segmentation (right). . . . 135
6.3 Neuro-symbolic architecture of a data science agent, illustrating the interaction between an LLM-based agent control loop, short- and long-term memory, and the execution environment. Dotted arrows denote neural–symbolic interactions. Solid arrows denote control and execution flows. Inspired by [Sun et al., 2025, Bhuyan et al., 2024] . . . 139
6.4 A KG-enhanced data science agent, extended from Zhang et al. [2023b]. . . . 140
6.5 Workflow for instantiating long-term symbolic memory in a data science agent. . 142

7.1 SEDAR's architecture . . . 150
7.2 Representative queries for ingestion, ML, and semantic data management. . . . 152
7.3 Workflow of the multi-agent system. Solid arrows denote core task flow, dashed conditional paths. . . . 153
7.4 Execution loop. Green=agents, blue=Python functions. Dashed/dotted edges indicate conditional flow/RAG. . . . 154
7.5 LangSmith trace for *"Get all datasets"* . . . 155
7.6 The full retrieval pipeline. The left side shows the indexing of the API functions and the right side the retrieval process including the re-ranking and caching. . . . 157
7.7 The new chat interface integrated into the frontend. The first step involves user confirmation for the predicted semantic labels of a dataset. . . . 159

8.1 Comparison of the height profile of craterings and pinholes. . . . 175
8.2 Output images of the automation script. . . . 176

8.3 Accuracy of a classification model [Doersch et al., 2020] plotted against the number of available samples (shots) when evaluated by a classification model. 176
8.4 Segmentation images and mask: (a) original image (b) cropped image (c) manually labelled image 179
8.5 Schematic diagram of the workflow from left to right 180
8.6 Left: Needle force in x-direction (y-axis) against the time (x-axis) for one particular sample. Right: optical image of the sample. 184
8.7 Left: Needle force in x-direction (y-axis) against the x-position (x-axis) for one particular sample and scratch. Right: Visualization of the scratch obtained from a laser scanning microscope. The green bar indicates the point of transitioning from the coated surface to the underlying substrate. 184
8.8 Example prompt using *JupyterAI* with *GPT*4's response to generate the necessary code for solving the image segmentation task. Shortened for brevity. 186
8.9 A report with two model recommendations generated by AssistML [Zacarias et al., 2023] for the tabular regression dataset. 191
8.10 End-to-end system combining semantic modeling and LLM prompting for automated code generation 192
8.11 Visualization of the ontology underlying the semantic model 194
8.12 Prompts for End-To-End ML 195

A.1 Images of the HTF equipment 253
A.2 Automated scratch test equipment 254

B.1 Ingestion workflow as depicted by Martin [2022] 256
B.2 Entry pages of SEDAR's UI. 258
B.3 The four steps in the dialog of the *Ingestion* tab 258
B.4 *General* Tab of a *DataSet*'s view 259
B.5 *Profile* Tab of a *DataSet*'s view 259
B.6 Illustrations of the *Lineage* Tab in SEDAR. 259
B.7 Search and ontology exploration functionalities in SEDAR. 260
B.8 General view of the *ML* tab 260

C.1 Combined semantic model with initial annotations on the primary keys (top) and finalized semantic model contextualizing three datasets (bottom). 266

D.1 Proposed ingestion pipeline 267
D.2 Proposed backend for generating reports 267

LIST OF FIGURES

List of Tables

3.1 Evaluation Dimensions for Data Architectures on a scale from 1 to 5, mapped to General Requirements . . . 63
3.2 Evaluation of Data Architectures against requirements . . . 64
3.3 Evaluation of real-world data architecture systems based on technical requirements. . . . 66

5.1 Examples of extracted key–value observations across source formats from the running example in figure 5.2. . . . 94
5.2 The pros and cons of LLMs vs KGs as described by [Pan et al., 2024] . . . 105
5.3 Comparison of semantic modeling platforms. . . . 111
5.4 The table shows the results of the four experiments that we run on three datasets. The datasets refer to the following original datasets from VC-SLAM [Burgdorf et al., 2022]: 1: 0001, 2: 0018, 3: 0068 . . . 121

6.1 Comparison of Search Engines, Knowledge Graphs, and LLMs across different dimensions relevant for automating data science, adopted from Hogan et al. [2025]. 146

7.1 Agent overview. . . . 153
7.2 Finetuning configurations and results . . . 161
7.3 Model Performance . . . 162
7.4 Node (**N**) & Tool (**T**) metrics across all models. Metrics marked with ↑ are better when higher; ↓ are better when lower. . . . 163
7.5 Task decomposition (shortened) for the query *"Predict failures in the Sensor_Data"* and retrieved functions . . . 168

8.1 Overview of the experiments related to object detection. Only *AutoGluon* can perform object detection. The open-source models were unable to deal with this task, hence *GPT*4 had to be involved. Distribution of train/test sets: 140/36 . . . 188
8.2 Overview of the experiments in the segmentation data set. Only *AutoGluon* can perform image segmentation. Distribution of train/val/test sets: 174/35/8 . . . 188
8.3 Overview of the image classification experiments with limited training data. *AutoGluon*'s few-shot setting leads to the best results. Distribution of train/test sets: first row 20/79 and second row 80/79 . . . 189

8.4 Overview of the regression experiments (1 & 2) on tabular data. Distribution of train/val/test sets: 204/10/41 190
8.5 Evaluating the data wrangling task for various LLMs 198
8.6 Evaluating the ML task for various LLMs 198

C.1 Natural-language glosses for the IRI identifiers used in the running example (Wikidata, RDF/RDFS/OWL/SKOS, XSD, and RML/R2RML). 261

E.1 Average cosine similarity scores for different query types each extended by four semantically distinct rephrasings 270

Listings

2.1 JSON document as an example for a nested data structure . . . 19
2.2 Examples of RDF literals . . . 27
2.3 Example RDF graph serialized in Turtle format, illustrating key concepts in the context of Earth orbiting the Sun via the WikiData free and open knowledge base 29
2.4 Example SPARQL query retrieving all planets that orbit stars, based on the RDF graph data stored in the WikiData knowledge base . . . 31
4.1 Creating a workspace using the SEDAR-API . . . 87
5.1 Example mappings extract from PLASMA for the Captials dataset . . . 125
5.2 RML mapping for the Captials dataset . . . 125
7.1 The main state. . . . 153
7.2 The state of the Execution loop. . . . 154
8.1 Example Triples from the Semantic Model . . . 194
C.1 RML Mapping for the running example. . . . 263
F.1 Target code for wrangling the data set in Python . . . 271
F.2 Crafting the ML model, imports truncated . . . 272

List of Abbreviations

HINT: In most PDF viewers, e.g. Acrobat Reader by Adobe, on your keyboard you can use `Alt + ←` to return after following a hyperlink.

AI Artificial Intelligence
ML Machine Learning
HSNR Hochschule Niederrhein University of Applied Sciences
HIT Institute for Surface Technology of the HSNR
i2dach Intelligente Infrastruktur für das Daten-Management in Automatisierten Chemischen Hochdurchsatz-Prozessen
DMBOK Data Management Body of Knowledge
DATER Defining Data Architectures and their Technical Requirements
DBMS Database Management System
FAIR Findable, Accessible, Interoperable, and Re-usable
URL Uniform Resource Locator
IRI Internationalized Resource Identifier
LLM Large Language Model
URI Universal Resource Identifier
Turtle Terse RDF Triple Language
SPARQL SPARQL Protocol And RDF Query Language
HTTP Hypertext Transfer Protocol
SME Small and Medium-sized Enterprises
HTF High-Throughput Formulation
ELT The Extract-Load-Transform principle
ETL The Extract-Transform-Load principle
MLOps Machine Learning Operations
OBDM Ontology-based data management
OBDA Ontology-based data access
OBDI Ontology-based data integration
SDM Semantic data management
KG Knowledge Graph
CST Chemspeed Technologies
RDF The Resource Description Framework

List of Abbreviations

RDFS RDF Schema
OWL The Web Ontology Language
SEDAR Semantic Data Reservoir
PLASMA Platform for Auxiliary Semantic Modeling Approaches
IDS International Data Space
YAML Yet Another Markup Language
RAG Retrieval Augmented Generation
GPT Generative Pre-Trained Transformer

1

Introduction

"There are decades where nothing happens; and there are weeks where decades happen."

Attributed to Wladimir Iljitsch Uljanow;
paraphrasing themes from [Lenin, 1975]

Artificial intelligence (AI), once the domain of science fiction, has now become a reality. In 1950, Alan Turing proposed a simple yet profound question: *Can machines think?* [Turing, 1950] His benchmark, the *Turing Test*, challenged a machine to imitate human conversation well enough to be indistinguishable from a person, which "has unquestionably generated more commentary and controversy than any other article in the field of artificial intelligence" [French, 2000]. For decades, this test remained an elusive goal, yet today, modern AI systems not only pass the Turing Test with ease, but far exceed its original intent. Modern AI systems now perform tasks once thought to be uniquely human: from pattern recognition and decision-making to complex reasoning and creative writing, composing music, and writing poetry. Powered by algorithms and fueled by vast amounts of data, AI has become indispensable in extracting insights from the ever-growing landscape of big data, driving innovation at a pace that once required decades of development to mere months or even weeks.

At the heart of this transformation is data. Without massive volumes of structured and unstructured data, today's AI systems would lack the material they need to learn, adapt, and improve. The discipline that underpins the systematic extraction of knowledge from data is known as data science. Combining elements of statistics, computer science, and domain expertise, data science provides the theoretical and practical foundations for building intelligent systems that can learn from data and generate actionable insights.

Data scientists are responsible for transforming raw data into meaningful insights that drive

decision-making. Machine learning (ML) is a core subfield of AI, focused on developing algorithms that enable systems to learn from data and make decisions with minimal human intervention. ML models improve their performance over time through continuous exposure to data, leading to more accurate predictions and classifications. In industrial contexts, ML is applied to optimize processes, predict maintenance needs, personalize customer experiences, and improve product quality.

As data continues to grow in volume and complexity, a phenomenon also known as big data, traditional data storage and management systems are increasingly strained. Here, the concept of a data lake emerged as a modern solution, designed to handle large, diverse datasets in their raw or unstructured form. Unlike traditional databases, which store data in structured tables, data lakes offer a flexible approach, allowing different types of data to coexist. This flexibility is essential in industries that rely on high-throughput data collection, such as chemistry, where data lakes support the storage and management of experimental data, metadata, and process information in a centralized repository.

This dissertation explores the integration of the aforementioned concepts—big data, AI, data science, ML, and data lakes—within the context of industrial and data-driven chemistry. It aims to address challenges in data management and to develop a data-driven framework that leverages a data lake to improve accessibility, analysis, and modeling of data in a chemical production environment.

1.1 Research Environment and Motivation

This research is situated within the domain of chemistry, with a specific focus on the paints and coatings industry. The digital transformation of industry, often referred to as Industry 4.0 [Ustundag and Cevikcan, 2018], is increasingly influencing various sectors, and the chemical industry is no exception.

Although chemical production is characterized by a high degree of automation, the development of new paint or dye formulations primarily relies on manual methods [Schmitz et al., 2022]. Experts typically propose experimental series (statistical design of experiments), often involving several hundred formulations that are mixed (formulation), applied to a substrate (application), and evaluated for desired paint or coating properties (e.g., scratch resistance [Steinmann et al., 1987, Randall, 2019]) using various measurement techniques (characterization). The interpretation of the measurement data is carried out by an expert, who then, if necessary, suggests further formulations, and this optimization process is usually repeated several times until a satisfactory result is reached. Despite extensive experience, human experts can only partially recognize the complex relationships between input factors (raw materials, process parameters) and measurement data [Mirabedini et al., 2012, Zareanshahraki and Mannari, 2021]. This problem is exacerbated when experienced experts leave the workforce, resulting in a loss of expertise [Kern et al., 2019, Quix et al., 2023].

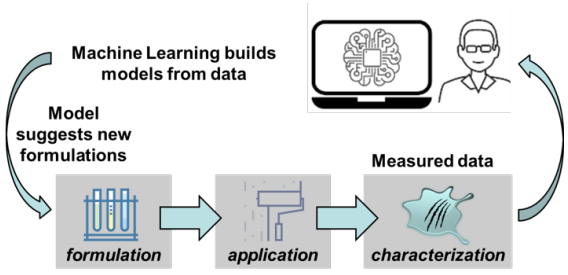
In particular, the paints and coatings industry faces unique challenges coming from reliance on complex formulation processes, stringent regulatory standards, and the need for rapid

innovation to meet environmental and market demands [Landwehr-Zloch and Glaß, 2020]. The sector is characterized by small and medium-sized enterprises (SMEs). Many of these companies are currently in a phase of transition and adaptation due to increasing demand for environmentally friendly products and intense international competition, which is particularly challenging for innovative SMEs as development times can be long. As in many other sectors, digitalization is considered a key factor in the chemical industry's transition to a sustainable and competitive future, and this concept is referred to as Chemistry 4.0 [Strehmel et al., 2019, Deloitte GmbH, 2017]. Automated systems for the formulation, application, and characterization of paints and coatings significantly shorten development processes. However, SMEs are hesitant to adopt such systems not only due to the high investment costs, but also because the control of these systems and the analysis of the generated data require advanced digital competencies [Floyd, 2021, Kallmuenzer et al., 2024].

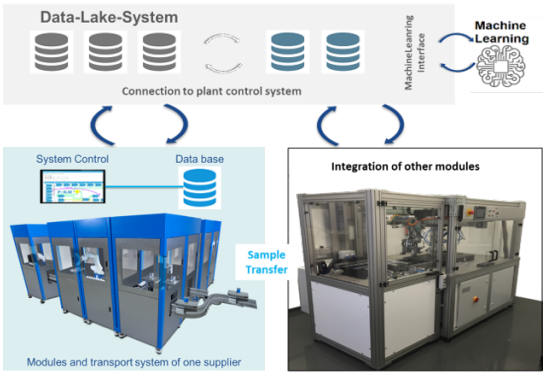
To address these challenges, the German Federal Ministry for Education and Research as part of the "Forschung an Fachhochschulen in Kooperation mit Unternehmen" (FH-Kooperativ) initiative funded the **I**ntelligent **I**nfrastructure for **D**ata Management in **A**utomated **C**hemical **H**igh-throughput Processes (i²DACH) research project at the Hochschule Niederrhein University of Applied Sciences (HSNR) [Quix et al., 2023]. The HSNR has a strong historical connection with the chemical industry, focusing heavily on paint chemistry and surface-related technologies, and the i²DACH research project builds on prior work conducted in the field of high-throughput formulation (HTF). Furthermore, the project actively involves two industrial partners from the region, integrating their products and expertise into the operational workflow. This collaboration not only strengthens the practical relevance of the research, but also ensures that industry-specific challenges are addressed effectively.

HTF enables fully automated material preparation and testing, facilitating faster screening of formulations for organic coatings and paints [Nandy et al., 2021, Schuchmann et al., 2021], while significantly improving the quality of experimental execution (e.g., by making measurement results more objective). At the HIT (Hochschule Niederrhein Institute for Surface Technology), a research-intensive institute of the HSNR that has *"Functional Surfaces"* as a key research focus, a modular HTF system (see Appendix A.1) from a private manufacturer is available, automating the essential process steps of paint development in collaboration with SMEs. Comparable systems are used in the chemical industry for formulation and raw material development [EVONIK, 2017]. The goal of this interdisciplinary project is the further development of the system into a so-called *"Smart Paint Factory"* [Stalmach, 2019], by integrating material development processes with automated control and data acquisition. Experimental data will be collected to create a comprehensive knowledge base, which will then be used for accelerated paint and coating development through data-driven modeling (see Figure 1.1a). ML models will iteratively create relationships between the elements of the formulation (raw materials, process parameters) and the target properties of the end product (e.g., physical properties like scratch resistance).

In this context, the vision of so-called *digital twins* [Singh et al., 2021] is particularly attractive. A digital twin refers to a virtual model of a physical process, product, or system, which



(a) Material development through iterative model creation [Quix et al., 2023]



(b) Planned architecture around the HTF system including a data lake and ML methods [Quix et al., 2023]

Figure 1.1: Proposed research in the i²DACH-project [Quix et al., 2023]

enables real-time simulation, monitoring, and optimization of development processes. For paint formulation, digital twins can replicate the formulation, application, and characterization stages, offering predictive insights based on real-time data reducing reliance on manual trial-and-error approaches [van der Aalst et al., 2023]. Braun et al. [2023] point out that a "Digital Twin of a [cyber-physical production [Uhlemann et al., 2017]] system consists of a set of models of the system, a set of Digital Shadows, and a set of services that allow using the data and models purposefully with respect to the original system." A so-called *Digital Shadow* is "a set of temporal data traces and/or their aggregation and abstraction collected concerning a system for a specific purpose with respect to the original system" [Braun et al., 2023] and thus, it can be viewed as a set of data observed from the system.

At the start of the i²DACH-project, neither the model building nor the system parameterization for a new formulation has been automated or integrated into the process flow. Continuous expansions with new functional modules are guided by the needs of the partners but pose significant challenges in terms of hardware interfaces, data management, and system control adaptation. Due to the modular nature of the system, data management is not fully integrated, and the adaptation of automated processes to new requirements must be performed manually [Hoseini et al., 2024e]. Data management was handled through proprietary solutions, meaning access to data is usually only possible through specialized control software and file-based exports. This approach, which is used not only here but also in other areas of chemistry, presents a hurdle for the further application of data science methods in the chemical industry [Beck et al., 2016].

The challenges of data management are to be addressed through the use of a *data lake* [Quix and Hai, 2019] (see Figure 1.1b). A data lake is a scalable data management platform that allows flexible handling of heterogeneous data and prepares it for data analysis. Due to the modular structure of the HTF system and its processes, various data in different formats are generated. i²DACH additionally emphasizes the use and evaluation of the data for experimental design and control. The ML methods will be integrated into the overall system of i²DACH,

enabling process control, data management, and analysis to be carried out on a common scalable infrastructure.

The project has the following research and development goals:

1. Development of standardized procedures for the formulation, application, and characterization of paints and coatings, and their transfer to the HTF system.
2. Automation of the system across multiple process levels through the integration of ML methods into the control layer of the HTF system.
3. Development of a data lake system to manage heterogeneous process data and efficient ML methods for analyzing this data.

The interdisciplinary nature of the project, combining chemistry, automation, and data science, supports three PhD candidates, each specializing in one of these fields. This dissertation is the outcome of one of these applied research efforts focusing on the **data science** part.

On the one hand, the project aims to perform the actual modelling using ML algorithms of chemical materials to predict material properties. This involves building models for coating formulations based on raw material characteristics to predict optimal raw material combinations and assess their influence on final product properties.

On the other hand, the doctoral research project focused on developing a data management platform for industrial and data-driven Chemistry, targeting the integration of chemical process data into a data lake system. This involves automating the time-consuming data preparation process, metadata management, query interfaces, data quality, and scaling for large datasets, enabling data to be enriched and prepared for ML applications. A key challenge is linking process data along the entire workflow, particularly for the HTF system. Hence the bulk of this dissertation is concered with the preliminary stages of any ML project prior to modeling.

1.2 Research Questions, Contributions and Outline

In pursuit of effective management of data within the context of the data-driven chemistry within the i²DACH research project, we consider four core research questions:

RQ1 — What constitutes a structured framework for characterizing and differentiating contemporary data architectures, and how do existing architectures align with the technical requirements articulated by this framework?

RQ2 — What kind of data-management system can fulfil the technical and domain-specific requirements of industrial chemistry, and what challenges arise when integrating heterogeneous data in such environments?

RQ3 — How to use semantics to improve the integration and usability of heterogeneous data generated in an industrial context, particularly facilitated within a semantic data lake?

RQ4 — How can a data management system empower non-technical users to perform complex data management tasks, such as integrating and analyzing heterogeneous data originating from industrial chemistry processes?

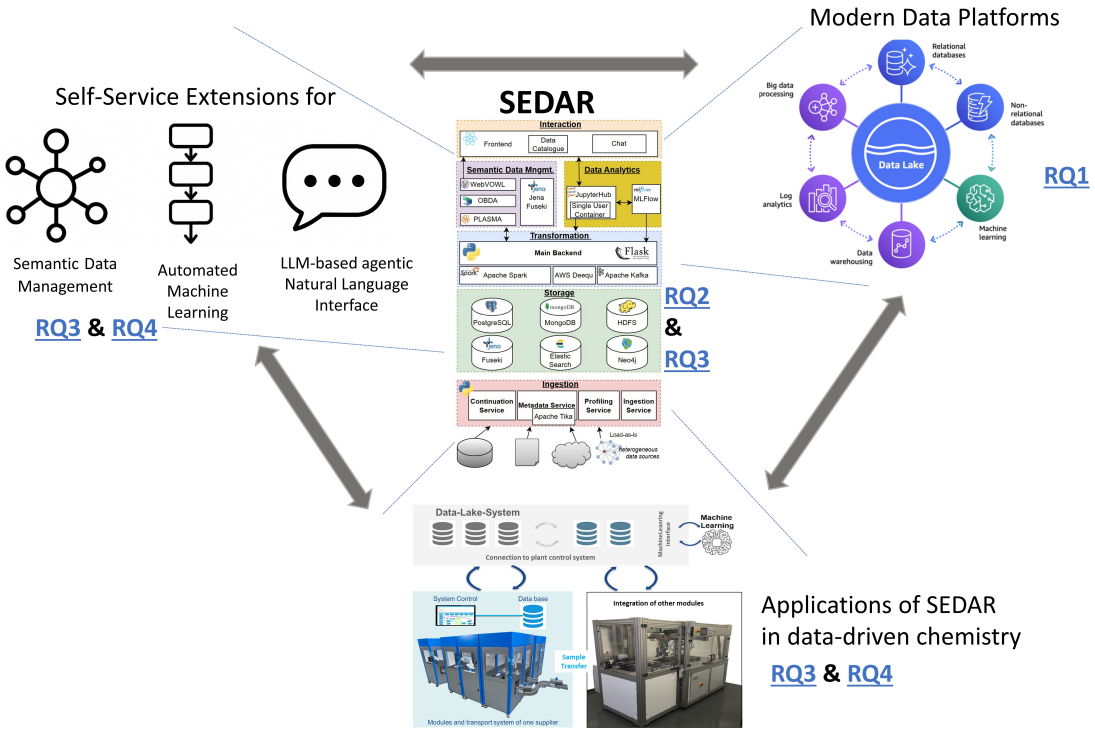


Figure 1.2: Various aspects in this work.

To answer the research questions formulated above, this work is structured as follows:

- Chapter 1 introduce the context of this work, motivates the research, and presents the research questions for the remainder of this work.
- Chapter 2 explores the theoretical and practical foundations of semantic data management, addressing RQ3.
- Chapter 3 addresses RQ1 and presents *Data Architectures and their technical Requrirements* (DATER), a structured framework for defining echnical requirements of seven predominant data architectures including an evaluation of real-world data management platforms.
- Chapter 4 addresses RQ2 and describes the core functionality of the *Semantic Data Reservoir* (SEDAR).
- Chapter 5 addresses RQ3 by presenting the semantic data management functionality of SEDAR .
- Chapter 6 addresses RQ4 by presenting recent approaches for data analytics implemented within SEDAR .
- Chapter 7 addresses RQ4 through a natural language interface for the SEDAR data management platform to lower the technical barrier for interacting with the system.
- Chapter 8 addresses RQ2 to RQ4 through concrete applications of data-driven chemistry conducted via SEDAR with an explicit focus on standardized data pipelines.
- Chapter 9 concludes the work, lists contributions, discusses its limitations, and motivates future research directions.

As illustrated in figure 1.2, the figure provides an overview of the central themes addressed in this dissertation. At the core lies the SEDAR system, which is the main contribution serving as the foundation for managing heterogeneous data. On the top right, the figure situates SEDAR in the broader landscape of modern data platforms (RQ1). On the bottom, it highlights the applications of SEDAR in data-driven chemistry (RQ2 & RQ4), demonstrating its practical integration into laboratory and industrial environments. Finally, the top left emphasizes self-service extensions to SEDAR facilitating semantic data management, automated machine learning, and a multi-agent natural language interface (RQ3 & RQ4). Together, these perspectives reflect how the research questions are addressed through the conceptualization, implementation, and evaluation of SEDAR in industrial chemistry.

2

Theoretical Background

"It is the theory which decides what we can observe."

Albert Einstein

This chapter lays the foundation for the dissertation by presenting relevant theoretical background. As Einstein's quote suggests, the way we understand and interpret data is shaped by the conceptual frameworks we adopt. In the context of this work, theoretical foundations not only inform our technical approach, but also determine which challenges we recognize and which solutions we make possible.

Accordingly, this chapter surveys core concepts in data and metadata management, governance, integration, quality assurance, analytics, and semantic technologies. These foundations prepare the ground for the subsequent chapters and they establish the conceptual vocabulary necessary to answer the research questions from section 1.2.

Chapter Outline

This chapter introduces the theoretical foundations required to understand the design, implementation, and evaluation of SEDAR. It is organized as follows:

- Section 2.1 — Clarifies the concept of data, including types, structures, lifecycle aspects, and the distinction between data, information, and knowledge.
- Section 2.2 — Provides an overview of core principles and tasks in data management, forming the conceptual basis for subsequent chapters.
- Section 2.3 — Introduces metadata categories, metadata models, and the role of metadata in supporting discoverability, governance, and automation.
- Section 2.4 — Summarizes the FAIR data principles and discusses their implications for system and architecture design.
- Section 2.5 — Presents frameworks and organizational considerations for governing data assets across their lifecycle.
- Section 2.6 — Describes foundational methods for integrating heterogeneous data sources, including schema mappings, ETL/ELT paradigms, and semantic integration.
- Section 2.7 — Covers key aspects of data quality management, including:
- Section 2.8 — Introduces principles of data analytics, including classical methods and modern machine-learning workflows.
- Section 2.9 — Provides a conceptual introduction to semantic data management, including ontologies, knowledge graphs, and semantic integration, serving as a foundation for later chapters.

2.1 Data

Understanding the fundamental concepts of data, information, and knowledge as well as their interrelationships is critical for any data-centric research.

The distinction between data and information has long been a subject of theoretical debate. In its most basic sense, this PhD thesis describes data as raw, unprocessed facts or observations, often lacking inherent meaning without context [International, 2017]. Information emerges when data is processed, structured, or interpreted within a context to yield meaning. This pragmatic definition avoids engaging in deeper philosophical debates about the ontological nature of data and information, as numerous competing definitions and frameworks exist in a range of fields. Physicists such as Ludwig Boltzmann [Boltzmann, 1877] formalized the concept of entropy in the context of thermodynamics and statistical mechanics, where it serves as a quantitative measure of disorder or uncertainty within a physical system. This interpretation implicitly links physical entropy to the concept of data as unresolved possibilities or hidden structure in the underlying system. Shannon's groundbreaking work in 1948 [Shannon, 1948] conceptualized information in terms of entropy in communication theory, framing it as the reduction of uncertainty in a signal. In this interpretation, a binary digit (bit) conveys the smallest unit of information—representing a choice between two possibilities.

In 1989, Ackoff proposed a more hierarchical view of cognitive structures, formalized in the *Data–Information–Knowledge–Wisdom pyramid* [Ackoff, 1989, Rowley, 2007]. According to this model, data becomes information when contextualized, information becomes knowledge when internalized and understood, and knowledge leads to wisdom through judgment and experience. This view emphasizes that information is not intrinsic to data, but emerges from interpretation emphasizing the need for so-called metadata, which is major concern in this thesis with an in-depth introduction in section 2.3.

Knowledge refers to the organized, internalized understanding that results from the interpretation of information. Unlike information, which is transient and context-dependent, knowledge is durable and actionable. It includes the ability to apply information to solve problems, make decisions, or infer new insights. Within data science and information systems, knowledge is more practically framed as experience or empirical evidence to foster understanding derived from the accumulation or synthesis of information through analysis. Knowledge can be explicit documented and shareable or it can be tacit, residing in individuals' skills or intuition. One major theme in this thesis is how to serialize knowledge in machine-readable representation to connect raw data with its broader context (see section 2.9). From a systems perspective, converting data into knowledge involves multiple layers of abstraction, transformation, and reasoning. The process requires not only contextual data but also domain expertise, methodological rigor, and interpretative frameworks. In this way, knowledge generation is the endpoint of a value-added chain that begins with data collection. This reflected also in many of the visualization for data architectures (e.g., figures 3.3, 3.4 and 3.7) in chapter 3.

The **data lifecycle** describes the stages through which data passes, from its initial creation or acquisition to its eventual disposal. Understanding this lifecycle is essential for effective data governance, analytics, and long-term stewardship. Data lifecycle models have been proposed in various publications in slightly different ways [International, 2017, Michener and Jones, 2012, Theissen-Lipp, 2024, Burton and Treloar, 2009]. The *Data Management Body of Knowledge* (DMBOK) [International, 2017] defines the following phases:

Plan refers to defining the strategic goals, governance frameworks, and policies for managing data while aligning with business objectives.

Design & enable refers to the data structures, models, metadata, and architecture, as well as setting up the necessary infrastructure and tools to support data operations.

Create/obtain refers to generating data internally or acquiring it from external sources. This includes data capture, ingestion, and integration into systems.

Store/maintain refers to persisting data in appropriate systems while ensuring its quality, consistency, and security over time through cleansing, validation, and monitoring.

Use refers to providing access to data for users and systems to support analytics, operations, and decision-making.

Enhance refers to enriching data by adding context, linking related datasets, or improving quality and usability through transformation, annotation, or semantic integration.

Dispose refers to securely deleting, archiving, or anonymizing data that is no longer needed, in line with legal, regulatory, or business requirements.

As described in the Data Science Process Model (v1.1) "***data science*** *is a field of interdisciplinary expertise in which scientific procedures are used to (semi)automatically generate insights from conceivably complex data leveraging existing or newly developed analysis methods. The knowledge gained is subsequently utilized, taking into account the effects on society*" [Schulz et al., 2022]. At its core, data science involves the end-to-end process of data handling: from data acquisition and cleaning to modeling, validation, and communication of results. Modern data science tools often incorporate elements of ML (section 2.8), natural language processing (section 2.8.1), and data visualization.

Any data science project requires various areas of expertise spanned across a variety of roles. The term **data scientist** is typically used broadly for anyone involved in a data science project [Schulz et al., 2022]. In this general sense, data scientists are seen as responsible for handling all stages of such a project. However, in a narrower and commonly used sense, a data scientist refers to a specialist focused on the analytical component tasked with choosing and conducting suitable analytical methods and tools and interpreting the outcomes. In this narrower definition, their involvement in other parts of the project is mostly advisory, thus those areas are handled by different specialized roles.

In large and complex data science endeavors, the responsibilities of a data scientist are often distributed among several sub-roles of which the following were particularly important for this PhD thesis: **Data analysts** [Schulz et al., 2022] apply data-driven analytical techniques, statistical modeling, forecasting and data visualization methods. Their tasks significantly overlap with those of data scientists in the narrower sense, which is why data analyst is sometimes considered an older term for the same role.

Data engineers [Schulz et al., 2022] are responsible for acquiring, storing, cleaning, structuring, and distributing data. They are mainly involved in the early stages of a data science project and focus on its technical and infrastructural aspects. In some contexts, they may also be referred to as **data architects**. A specific sub-role under data engineering, especially common in larger projects, is the **data steward** (also called **data manager**). They focus on managing data access, ensuring data security, and maintaining high data quality making them closely connected to the technical application area.

Domain experts [Schulz et al., 2022] serve as key users or representatives thereof, possessing deep knowledge of the specific field in which the actual data emerges from. They help define relevant aspects to be modeled or analyzed and bridge the gap between technical and methodological experts. In this research, this role was represented by **coating chemists** (my fellow co-authors G. Zhang, C. Schmitz) as well as **automation experts** (my fellow co-authors D. Polke and A. Surjana), who collaboratively defined the specific goals and modeling spaces for the experimental design. The coatings chemists contributed their in-depth understanding of formulation principles, material interactions, and performance requirements relevant to the application, ensuring that the experiments focused on scientifically meaningful and industrially relevant parameters. Meanwhile, the automation experts played a key role in translating these objectives into a framework suitable for high-throughput experimentation, integrating robotic systems and data workflows to systematically explore large combinatorial spaces.

2.2 Data Management

In today's data-driven world, the efficient management of data is paramount for stakeholders across various domains. Managing large volumes of data poses significant challenges due to the variety of formats and data models, distribution across departments, and different governance structures and network topologies within organizations. In both research and industry environments, this complexity is compounded by the need for streamlined data handling processes to support recent technological developments that place data into the center of scientific discovery [Tenopir et al., 2020] as well as business models [Popovič et al., 2018].

Historically, data was viewed as a by-product of legacy transactional systems and held little value beyond supporting the applications that processed it [Al-Ruithe et al., 2019]. This mindset persisted until the early 1990s, when businesses began recognizing data's potential. Data-driven decisions and processes emerged, leading to greater investment in managing the increasing volume, velocity, and variety of data. Complex systems like data warehouses [Inmon, 2005], ERP [Davenport et al., 1998], and CRM [Payne and Frow, 2005] were developed to handle these challenges. As data links grew more intricate and shared across systems, the need for a single source of truth became essential, giving rise to **master data management** [Loshin, 2010], which refers to the management of **master data**. Master data is that data that is most critical to an organization or enterprise like for example customers, employees, vendors, suppliers, products, etc.

The continuous and overwhelming increase of available data called for a paradigm shift in the computing architecture and large scale data processing mechanisms. In 2007, this shift was called the "Fourth Paradigm: Data-Intensive Scientific Discovery" by Jim Gray, a database pioneer [Hey et al., 2009], with the first three paradigms being experimental, theoretical and, more recently, computational science. Gray argued that the only way to cope with this paradigm is to develop a new generation of computing tools to manage, visualize and analyze the data flood.

At the HIT, where this research takes place (see section 1.1), similar challenges need to be addressed [Beck et al., 2016]. There, an automation platform for material development is in operation and various datasets of experiments for paints and coatings need to be captured, cataloged, and made available for ML to be applied in chemistry for experiment suggestions and analysis. Sources that generate data are very heterogeneous, e.g., streams of machine sensors, interfaces of control software, databases and images of microscopes, and scripts and model checkpoints from ML. As a result, tasks for analyzing data, such as collecting, accessing, searching, understanding and processing data, become very time-consuming. This makes it difficult to realize visions such as **Chemistry 4.0** [Strehmel et al., 2019], which refers to the digital transformation of the chemical industry and emphasizes the integration of data-driven systems for increasing degrees of automation.

To tackle this challenge the field of **data management** has attracted attention which is the practice of collecting, processing and using data securely and efficiently [Silberschatz et al., 2002]. Another definition is given in the DMBOK [International, 2017]: "Data Management is

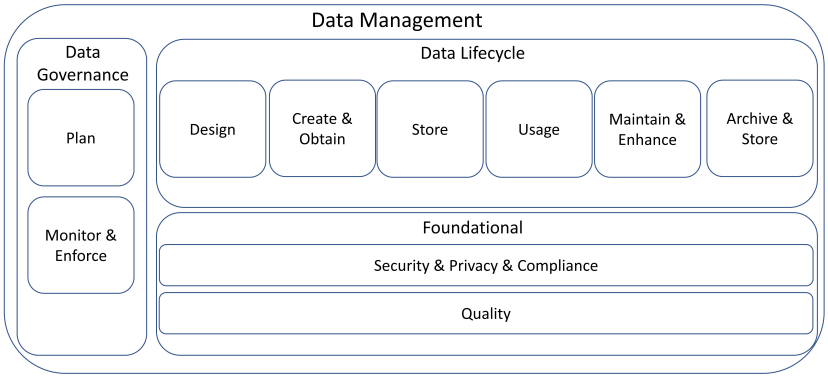


Figure 2.1: Data Management Activities as depicted by [Eichler et al., 2021] and based on the *DMBOK* [International, 2017].

the development, execution, and supervision of plans, policies, programs, and practices that deliver, control, protect, and enhance the value of data and information assets throughout their lifecycles". Figure 2.1 gives a rough overview of activities in the field of data management. Data governance (see section 2.5) presents the basis for data management as it involves the planning, monitoring and enforcement of both the data lifecycle management and foundational activities [International, 2017]. The **Data Lifecycle**, as previously described, represents the stages through which data passes during its existence, design, creation, obtaining, storing, using, maintaining, enhancing as well as archiving and deleting of data [International, 2017, Yu and Wen, 2010]. The foundational tasks are performed throughout all of the above described lifecycle steps and include the realization of security, privacy, compliance and data quality management.

This work specifically focuses on data management within the context of **database management**, which involves the structured organization, storage, and retrieval of data within databases [Ramakrishnan and Gehrke, 2003]. Traditionally, databases are thought of as centralized systems for storing data assets. However, recent technological advancements have dramatically transformed this landscape, introducing a wide variety of new data architectures including distributed databases and cloud-based systems [Özsu et al., 1999, Wang et al., 2020, Nargesian et al., 2019]. As a result, the number and diversity of data architectures have increased immensely, offering unprecedented flexibility and scalability in managing complex data ecosystems. Some of these architectures are discussed in chapter 3. Data management is also relevant in other areas where a conventional database might not be useful, because for example data is volatile and processed on-the-fly (e.g., in-memory computing, data stream processing [Carbone et al., 2015]).

What follows is relevant background and related work in the field introducing the main concepts relevant for data management in the context of scalable data architectures. We begin with metadata management (section 2.3), data governance (section 2.5), data integration (section 2.6), data quality management (section 2.7) and data analytics (section 2.8) followed by a discussion of semantic data management (section 2.9) that starts with an introduction into the fundamentals of Semantic Web technologies (section 2.9).

2.3 Metadata management

The mere collection of datasets is not sufficient. Without context, data itself has no meaning or value because it is without context and interpretation. With context, information can be inferred from data, and only then it delivers meaning and value to the recipient [Rowley, 2007, Groff and Jones, 2012]. For data to be useful, it must thus be accompanied by meaningful **metadata**, i.e., data about data [Marco, 2000] or data that describes other data. This means metadata are also data, and data become metadata when they are used to describing other data [International Organization for Standardization, 2023]. The importance of metadata is rooted in the fact that those users, who publish or provide dataset and are responsible for the data, may not belong to the group, who are going to use the data later on. Thus, the **implicit context knowledge** needs to be committed alongside any created artifacts to allow a third party to interpret the received assets later on. This is especially crucial for recipients with limited knowledge in the domain from the data arises or those unfamiliar with the specific data sets. Without such metadata, accurately interpreting the data becomes a major challenge [Jain and Sinha, 2010], hence in order to conduct data management, metadata is required throughout the depicted tasks in figure 2.1.

A basic example is known to many from school, where the teacher asks the student: "How far did you run in the race?" The student responds with: "500." The teacher, sensing the ambiguity, replies, "500 what? Meters? Kilometers? Steps?" The student then clarifies, "Oh! 500 meters." The initial response, "500," is a raw number without context, making it unclear which unit of measurement is meant. Only when the unit "meters" is provided does the data become meaningful. For another example we consider a particular tabular dataset where three individual columns in the table may contain arbitrary numeric data, and without further context, there is no way of interpreting what the numbers represent and what their relationships are. However if we observe the column's names, for example "Product ID", "Quantity sold" and "Total Sales (€)", then we can derive the information that we are likely looking at a table about the frequency and revenue of products. The set of all columns of a table, also referred to as its **schema** is a typical and most basic form of metadata.

The use of metadata, along with the actual data, is a core research topic in database management technology [Sawadogo et al., 2019, Edara and Pasumansky, 2021, Scholly et al., 2021, Quix et al., 2016, Zhao et al., 2021b]. It has emerged as early as the 1960s with the implementation of file systems and B+-trees, which store field names of the individual data records in a database along with their data types and these are essentially metadata [Sen, 2004]. **Metadata management** is the task of acquiring, storing, utilizing, and enriching metadata automatically whenever possible hence "metadata management is data management for metadata" [Eichler et al., 2021]. This is a complex problem, and this work has a special focus on this task in the context of data architectures, i.e., provide context about the various data assets to assist data-intensive applications. For instance, ensuring data privacy/security requires defining and enforcing access rights, which are a form of metadata, as they specify who is permitted with access. Similarly, metadata is used to document data quality by capturing

information such as completeness. Each data management task involves multiple use cases that necessitate the collection of metadata. For example, the "Usage" task may require tracking data lineage, monitoring access patterns, and more.

Metadata categories

Organizations need to have metadata related to their data sources in order to have an accurate inventory of data files, their origins, and their value. An early categorization is introduced by Quix et al. [2016] focusing on (semi-)structured data where metadata is classified into the three groups: (1) **structural data**, containing information about the schema and the data type of individual datasets, (2) **metadata properties**, i.e., a set of properties for each data source, e.g., filename, size, location or date of last modification, the date or the author of an experiment, and (3) **semantic data** handling the meaning of the entities by allowing annotations, e.g., via a URI (Universal Resource Identifier) that refers to some ontology term (see section 2.9).

Diamantini et al. [2018] define three categories for metadata: **business**, **operational**, and **technical** metadata. Business metadata are defined as the set of descriptions that make the data more understandable, i.e., they provide semantic context. Such metadata are mostly entered manually by the data provisioner. Operational metadata are information automatically generated during data processing like data location, file size, number of records, etc., as well as process information. Technical metadata express how data are represented, including data format (e.g., raw text, JPEG image, JSON document, etc.), structure or schema. In this model, business, operational and technical metadata sometimes intersect, e.g., data formats may be considered as both technical and operational metadata.

Sawadogo and Darmont [2021] describe metadata categories classified with respect to structural metadata types. **Intra-object metadata** are associated with individual atomic data assets ("*objects*"), e.g., a CSV file, an image dataset or a video file. They are subdivided into four main subcategories:

1. properties provide an object's general description, e.g., file name and size, location, date of last modification, etc.
2. visualization and summary metadata aim to provide an overview of the content or structure of an object, e.g., row counts, averages, table schemas, or word clouds for textual data.
3. version and representation metadata are obtained through data updates, while representation metadata come from data refining operations. For instance, a refining operation may consist of vectorizing a textual document into a bag-of-words for further automatic processing.
4. semantic metadata involve annotations that describe the meaning of data in an object and are detailed in section 2.9.

Inter-object metadata encode relationships between data assets and are subdivided into three categories:

1. groupings organize data assets into collections and it is possible to deduce such links

automatically from some intra-object metadata,

2. similarity links express the strength of likeness measured against chosen metrics, and (3) parenthood links save the lineage, i.e., when a new data asset is derived from a combination of others. Finally, there are **global metadata**, i.e., data structures that provide a context layer to make data processing and analysis easier, hence they are not specific to an individual atomic data asset, e.g., logs, indexes or business glossaries.

2.4 The FAIR Data Principles

Wilkinson et al. initially proposed the FAIR principles in 2016 [Wilkinson et al., 2016]. By adhering to these data management practices, which essentially demand a standardized procedure to publish sufficient corresponding metadata alongside individual datasets, multi-source data can be more effectively shared, combined, and analyzed, fostering collaboration and reproducibility. The acronym FAIR expands to the following principles for data:

1. **Findable**: Data require globally unique and persistent identifiers and need to be described with rich metadata. Both of these resources are to be indexed in a searchable resource.
2. **Accessible**: All data has to be retrievable using their specific identifier. Standard communication protocols are to be used, which need to be open, free and universally implementable, supporting authentication and authorization procedures. Furthermore, metadata needs to be persisted to be available even if the described data ceases to be available.
3. **Interoperable**: All data and metadata need to use a shared and applicable language for knowledge representation. The vocabularies used in the description also need to follow the FAIR principles and metadata is expected to include references to other (meta)data.
4. **Re-usable**: Data as well as metadata need to be released stating a proper license, be associated with their provenance and meet domain-relevant community standards.

2.5 Data Governance

The term "governance" in general refers to the way an organization goes about ensuring "that strategies are set, monitored, and achieved" [Rau, 2004] and data governance can be viewed as "the specification of a set of information policies that reflect business needs and expectations, along with the processes for monitoring conformance to those information policies" [Loshin, 2010] or “a system of decision rights and accountabilities for information-related processes, executed according to agreed-upon models which describe who can take what actions with what information, and when, under what circumstances, using what methods” [The Data Governance Institute, 2024a]. Often researchers and practitioners confuse between data governance and data management [Al-Ruithe et al., 2019], because governance plays a core role (see Figure 2.1). However, from the definitions it is clear that data governance is only a subfield of data management as it is more concerned with policies to deal with the oversight and control of data. On the other hand, it does not deal with the technical details of how to store data physically for example, which is part of the larger area of data management.

At its core, data governance involves defining clear roles and responsibilities, such as data stewards [Wilkinson et al., 2016] and governance committees, who make and enforce decisions regarding data-related processes. These decisions typically cover data quality and access, but also security, privacy, and regulatory compliance. The goal is to create a consistent framework that supports cross-functional data use, aligning rules and monitoring compliance across the organization. This also includes the notion of 'long-term care' of valuable digital assets, with the goal that they should be discovered and re-used for downstream investigations, either alone, or in combination with newly generated data [Wilkinson et al., 2016].

Organizations typically formalize data governance when their size or complexity demands more structured approaches to managing cross-functional data processes, or when regulatory requirements necessitate stricter oversight. By following a governance framework, organizations can clearly define how data-related decisions are made, who is responsible, and how compliance is monitored [The Data Governance Institute, 2024b].

2.6 Data Integration

Data integration refers to the task of combining multiple heterogeneous data sources into a unified global representation for data access [Miller et al., 2001]. In industrial settings, vast amounts of data from heterogeneous sources are generated. At the HIT (see section 1.1) this includes, but is not limited to, binary sensor data, logs, images from microscopes, and semi-structured datasets from IoT devices. These data sources often have different formats: structured (tables, entire databases), semi-structured (like XML and JSON), and unstructured (like text, logs, images, and videos). In this context, data integration is vital for combining all this diverse data into a coherent, unified representation.

Literature on data integration traditionally involves combining structured data, which refers to the tabular/matrix representation of data, typically using predefined schemas [Özsu et al., 1999]. Integrating unstructured data like text, documents, images or videos presents additional challenges, as it lacks the rigid organization and clear relationships found in structured data. Semi-structured [Buneman, 1997] data models like XML and JSON, that got popularized through the web, offer more flexibility in terms of schema representation and data format. For data integration this flexibility requires more sophisticated techniques to ensure consistency across varying sources while preserving the hierarchical or nested relationships within the data. Listing 2.1 shows the structure of an exemplary nested JSON dataset containing information about two individuals which demonstrates these challenges. Semi-structured data serializes data by annotating each data item explicitly with its description, i.e.,so-called *key* ("name", "phone"), thus they are often explained as "self-describing" or "schema-less". One of the main strengths of semi-structured data model is that it can accommodate variations in its structure. However, this comes at the cost of missing data, duplicate fields or minor changes in representation for different data points. If we were to transform the dataset into a table, we immediately face the challenges of how to deal with the fact, that the second person's key "name" is nested, i.e., it contains the additional keys "first" and "last, which are not present for

```
{ person:
  {
    name: "Albert Einstein",
    phone: +49 3127786,
    email: "aeinstein@princeton.edu"},
  person: {
    name: {
      first: "Emmy",
      last: "Noether"
    },
    phone: +49 2136877,
    email: "enoether@math.uni-goettingen.de"  } }
```

Listing 2.1: JSON document as an example for a nested data structure

the first person. Handling such missing or incomplete schema definitions is a common obstacle in semi-structured data integration [Abiteboul et al., 2000].

Data integration for tabular data is classically achieved by defining a global schema, also referred to as mediated schema, which is obtained by merging all the source schemas against a global schema. This is usually performed in four stages [Hai, 2020]:

1. **schema matching** first discovers the schemas of the individual data sources, and then finds the relationships between them. Here, challenges emerge for example when the individual column names are cryptic or follow heterogeneous naming conventions. Similarities between individual schema entries can be automatically deduced through a variety of methods such as based attribute names and descriptions, constraints, data types, instance values [Rahm and Bernstein, 2001].
2. **schema merging** is the process of defining the global schema based on the information obtained from schema matching.
3. **schema mappings** are then generated based on the given source schemas, the global schema, and their correspondences.
4. **query reformulation** converts user queries posed on the global schema to a set of subqueries for the source schemas.

In this context, **data wrangling** [Rattenbury et al., 2017, Kandel et al., 2011] has emerged as a crucial process that goes beyond basic integration tasks (see section 2.7). It refers to the transformation and mapping of raw data into a more usable format, often involving more complex restructuring or enrichment processes. This step typically prepares the datasets for analysis by extracting data from heterogeneous data sources, integrating and cleaning them, standardizing formats and handling intricate data transformation needs to eventually storing them in aggregated form. Automated tools for data wrangling can significantly ease the process, enabling data professionals to tackle large-scale datasets more efficiently [Furche et al., 2016].

Furthermore, there is an important distinction to be considered. Typically, there are two ways of data integration, either following the **Extract, Load, Transform principle** (ELT) or following the **Extract, Transform, Load principle** (ETL) principle. ELT does not force the source data to be transformed prior to being loaded, i.e., it can be accessed on-demand, meaning schema definition, integration, indexing, etc. should be done only if necessary at the time of data access with so-called wrappers [Quix and Hai, 2019]. In contrast, ETL tools require the

source data to be first both extracted and transformed into the unified global schema, before they can be loaded into storage to then be accessed [Quix and Hai, 2019].

2.7 Data Quality Management

As Wang and Strong [1996] point out "high-quality data is essential because it enables accurate decision-making, supports efficient operations, and fosters trust among data consumers". An article by Forbes from 2016 found that data preparation activities consume up to about 79% of a data scientist's time, and up to 78% of data scientists declare them to be the least enjoyable part of data science [Press, 2016]. This highlights the critical yet time-consuming nature of maintaining high-quality data.

Weber et al. [2009] define data quality management (DQM) as "the quality-oriented management of data as an asset, [...] aimed at continuously improving data quality." Their work explores the relationship between data governance and DQM, demonstrating that organizations can establish accountability and culture for high data quality standards by involving both business and IT professionals through data governance policies.

Data or information quality itself is defined based on two key aspects: first, its dependence on the user's needs; and second, its "fitness for use" referring to the ability of data to meet the requirements for a specific use case [Wang and Strong, 1996, Redman, 2001]. Both these definitions agree on the fact that data quality is considered a multifaceted construct, consisting of a set of data quality attributes also referred to as data quality dimensions requiring consumer assessment [Wang and Strong, 1996, Loshin, 2010]. Examples for such dimensions are accuracy, completeness, consistency, relevancy, and timeliness, but depending on the specific use case or task at hand, there can be many more.

2.7.1 Data Profiling

The first step towards improving and sustaining high-quality data is to derive a **data profile**. A data profile is a comprehensive summary of the dataset that includes for example key metadata such as statistical distributions, data types, patterns, missing values, and potential relationships between columns of tables [Abedjan et al., 2019]. **Data profiling** is the process of analyzing datasets to discover such metadata. As the key challenge in data management is the lack of metadata for datasets, data profiling helps reconstruct this metadata, supporting tasks like data type identification, key dependencies, and data quality assessment. Data profiling is a crucial component of data preparation, which consumes a significant portion of a data professional's time. The complexity of data profiling grows with the size of the dataset, making scalability a significant concern. Three major challenges in data profiling include determining the right profiling tasks appropriate for the use case, efficiently processing large datasets, and interpreting the results. While data profiling tools may automate many of these tasks, expert interpretation is often required to make sense of the results and derive meaningful insights from the metadata.

2.7.2 Data Cleaning

With a data profile at hand the dataset's quality issues are identified. The next logical step in data quality management is **data cleaning**. Data cleaning refers to the process of correcting or removing inaccuracies, inconsistencies, and errors within a dataset to ensure its reliability and usability. This step is essential because even high-quality data profiles can reveal issues such as missing or invalid entries that need to be addressed before the data can be effectively analyzed. Some typical data cleaning steps involve the handling of missing data, removing duplicate entries, correcting inconsistencies, e.g., when data values are entered in different formats or units, and validating data, i.e.,to ensure that the data values meet expected constraints, such as valid ranges for numerical values or correct formatting for categorical data. Data cleaning is a time-intensive task, often requiring manual intervention or specialized tools [Kandel et al., 2011]. However, it is a crucial aspect of preparing data for analysis, as inaccurate or incomplete data can lead to flawed insights and poor decision-making.

Data cleaning plays a crucial role in the data science workflow [Polyzotis et al., 2017]. Clean data is essential for ensuring that models and analyses yield accurate and meaningful results. This concept is colloquially known as *"garbage in, garbage out"*, meaning that if the input data is flawed or inaccurate, the resulting analysis or model output will also be unreliable [Gottinger, 2017]. Without proper data cleaning, even the most sophisticated algorithms are likely to produce misleading insights. In fact, as noted earlier, data scientists spend a significant portion of their time on data preparation, which includes data cleaning, as it forms the foundation for all subsequent analysis.

In addition to improving data accuracy, data cleaning also supports data integration (see section 2.6) by ensuring that datasets from different sources can be combined seamlessly. Clean, standardized data is easier to merge and compare, allowing organizations to draw insights from multiple datasets effectively [Kimball and Caserta, 2004].

2.8 Data Analytics

Once data is cleaned, integrated, and structured, it can be leveraged to draw conclusions and gain insights. The term **data mining** first appears in the 1980s [Lovell, 1983, Runkler, 2020] and its goal is to extract knowledge from data [Fayyad et al., 1996]. Here, knowledge refers to all kinds of interesting patterns that are generally valid, novel, useful, and interpretable to humans [Runkler, 2020]. The relevance of these patterns is application-specific, and their significance often requires validation by domain experts. A canonical example includes the so-called apriori algorithm [Agrawal et al., 1993] with which retail companies mine transaction data to discover insights like for example customers who buy diapers often also purchase beer, particularly on weekends. This insight leads the store to strategically place these items together, increasing sales [Dongre et al., 2014].

The term **Business Intelligence** refers to a type of data analysis aimed at understanding organizational activities and opportunities to improve organizational success. Business Intel-

ligence went hand-in-hand with the emergence of so-called data warehouses [Inmon, 2005] (see section 3.3.1) and has the goal of asking the right questions about a businesses' data, so it can gain insights to make better decisions [International, 2017]. An example is the sales performance dashboard used by companies that visualizes sales data across different regions, product lines, and time periods. Management can track performance, compare regions, and make informed decisions about marketing or inventory adjustments [Inmon, 2005].

The term **data analytics** first appeared in the early 2000s [Tyagi, 2003, Kohavi et al., 2002], is distinguished from data mining as the application of computer systems to the analysis of large data sets for the support of decisions. It is a critical component ("Usage") of the broader data management lifecycle. There are different types of analytics used depending on the objectives, including descriptive analytics (summarizing historical data), predictive analytics (using data to forecast future trends), and prescriptive analytics (providing actionable recommendations based on the data), but also visualizations [International, 2017]. An example for predictive analytics can be found in manufacturing, where a company analyzes sensor data from machinery to predict equipment failures. By analyzing trends and deviations from normal operations, they schedule maintenance before breakdowns occur, thereby reducing downtime [Aivaliotis et al., 2019].

In the context of big data and integrated systems, data analytics often employs ML techniques. Data Management for ML has been well-researched for at least ten years [Chai et al., 2023]. To keep track of the various inputs and outputs of an ML application (e.g., data versions, code and tuning parameters, reproducing results, and production deployment), one subfield is proposed also known as **Machine Learning Operations** (MLOps) streamlining the ML lifecycle [Alla and Adari, 2021]. MLOps [Alla and Adari, 2021] encompasses many practices for deploying, maintaining, and managing ML models easily, efficiently, and reliably. Within an MLOps framework, data collection, analysis, and cleaning can be organized. Additionally, version control for data is provided, the model and underlying program code are stored, the ML model can be automatically deployed or tested, and it can be evaluated whether the model performs well and results are reproducible [Kreuzberger et al., 2022]. ML-driven metadata management includes multiple steps, such as model training, hyperparameter tuning, debugging, deployment, and more [Ashmore et al., 2021]. Apart from providing the relevant descriptive metadata of data sources in the lake, coverage of the metadata of ML models based on this data is thus required also [Hai et al., 2023, Zhao et al., 2021b, Schlegel and Sattler, 2023].

Modern data architectures are required to support analytics processes [Hai et al., 2023] to provide the information about connections between models and datasets. Here, typical queries include but are not limited to: "On which dataset D was model M trained on? What data preprocessing was necessary? Where are the model artifacts stored and who has access to them?" Zhao et al. and Schlegel et al. [Zhao et al., 2021b, Schlegel and Sattler, 2023] present metadata models to capture not only descriptive but also info about performed analyzes on top of stored datasets.

2.8.1 Large Language Models

In November 2022, *OpenAI*, a San Francisco-based tech company, unveiled *ChatGPT*, a web-based chatbot designed to respond to user queries with human-like fluency. Its public release quickly became a global phenomenon, attracting over 100 million monthly users by January 2023[1], earning it the title of the fastest-growing consumer application in history. ChatGPT captured the public's imagination with its conversational quality, its ability to provide in-depth responses to complex technical questions across numerous domains, its exceptional performance on standardized tests, and its built-in safeguards to restrict responses on controversial topics. Within months, other major players launched competing chatbot models, including Microsoft's Bing Chat (February 2023), Google's Bard (March 2023), Anthropic's Claude (March 2023), and Baidu's Ernie (August 2023) [Kamath et al., 2024]. While AI researchers familiar with recent advancements were less surprised by these breakthroughs, the release of *ChatGPT* marked the broader public's first encounter with the power of AI.

Around the same time, advancements in another domain emerged: text-to-image models. These systems enable users to input textual descriptions and generate digital images matching those descriptions. By 2022, state-of-the-art text-to-image models had reached a level of photorealistic quality, with their outputs often indistinguishable from genuine photographs. These models also demonstrated the ability to mimic artistic styles, including those of famous artists throughout history. Beyond text and image generation, related applications gained traction in 2022 and 2023, such as synthetic voice generation that mimics celebrities and deepfake technologies capable of altering live video appearances and speech. Recent advancements have extended capabilities to multimodal models, which can process and generate both text and images (e.g., DALL-E and CLIP). These models enable seamless integration across modalities, broadening their applications to areas like text-to-image generation, video analysis, and beyond. Despite the diverse applications of these technologies, they all share a common foundation: Large Language Models (LLM) [Kamath et al., 2024].

In text-to-image generation, LLMs do not generate images directly. Their role lies in their ability to process and understand natural language prompts, which are then used to guide image synthesis models. They play the critical role in ensuring that the input text is accurately interpreted and transformed into meaningful visual representations. LLMs are adept at parsing and understanding complex, nuanced textual inputs, in short semantic understanding. And it is exactly this ability that makes them so attractive also for this work. A major theme of this PhD thesis concerns understanding and generating the semantics and relationships between heterogeneous datasets. As we will see, here LLMs play an increasingly crucial role (see chapters 5 to 7).

Core Concepts of LLMs LLMs are built upon a series of advancements in ML, with the transformer architecture at their core, which were introduced by Vaswani et al. in 2017 [Vaswani,

[1] https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/, accessed on 02.05.2026

2017]. Its key innovation lies in the self-attention mechanism, which allows the model to focus on relevant parts of the input text, regardless of their position. This overcomes the limitations of earlier recurrent and convolutional neural networks, which struggled to capture long-range dependencies in text. This includes an Encoder-Decoder Structure. While some models like BERT use only the encoder and others like GPT rely solely on the decoder, both components enable the transformer to process input sequences efficiently [Zhou et al., 2024]. Transformers use positional embeddings to retain the order of words, ensuring that meaning is preserved in sequential data. Furthermore, the scalability of the transformer architecture makes it ideal for training on massive datasets, a hallmark of LLMs. Tokenization involves converting sentences into a machine-readable format by breaking them into smaller units, referred to as tokens. Depending on the required granularity for a specific application, this process can be performed at the level of words or subwords. In word-level tokenization, each word in a sentence is treated as an individual token. These tokens are subsequently mapped to vector representations, such as word embeddings, which serve as the input to the Transformer model.

LLMs typically undergo a two-step training process:

1. Pre-Training: During this phase, the model is trained on vast amounts of unlabeled text data to predict missing words or complete sentences.
2. Fine-Tuning: Once pre-trained, the model is adapted to specific tasks or domains using smaller, labeled datasets. Fine-tuning allows the LLM to specialize in tasks such as sentiment analysis, question answering, or domain-specific text generation, improving performance in targeted applications.

At the heart of LLMs lies their ability to encode text into high-dimensional vector spaces, known as embeddings. These embeddings capture the semantic relationships between words, phrases, or even entire documents. For example:

1. Words with similar meanings are located closer together in the embedding space.
2. Contextual embeddings, such as those produced by transformers, account for how the meaning of a word changes depending on its usage (e.g., "bank" as a financial institution versus "bank" of a river).

This semantic richness allows LLMs to process and generate text with nuanced understanding.

A defining feature of LLMs is their scale, measured in the number of parameters (i.e., trainable weights). Modern LLMs contain hundreds of billions of parameters. Scaling up the size of the model improves its ability to capture complex language patterns, albeit at the cost of increased computational resources and demand for training data. The computational and resource-intensive nature of training and inferencing LLMs is a major obstacle in 2025 for many researchers. For example, one of the largest and most powerful open-source LLMs at the time of writing this text, the LLamA 3.1[2] with 405 billion parameters requires 1944 GB of GPU memory in 32-bit mode. With one of NVIDIA's premium GPUs (A100, 80 GB GPU memory, $\sim$12.000€) for servers, this would require roughly $\sim$31 such GPUs for inference and 122 GPUs for training.

[2] https://ai.meta.com/blog/meta-llama-3-1/, accessed on 02.05.2026

2.9 Semantic Data Management

Semantic data can enrich basic metadata, such as schema and data types, by adding context information not originally present in the data source. One way to achieve this is to map raw data from heterogeneous data sources to semantically rich models to increase the usability and interpretability of data. In what follows, we introduce relevant background on the fundamentals of semantic technologies. We describe how to utilize these techniques for the management as well as access and integration of heterogeneous data sources (Section 5.2).

While we recognize the many ways the modern Web enhances our daily lives, we often overlook its limitations. Despite its successes, the Web is far from perfect, and understanding its shortcomings is key to improvement. While one might first think of issues like speed or accessibility, these relate more to the underlying Internet than the Web itself. A significant limitation lies in the Web's inability to easily handle tasks where information is spread across multiple sites [Hogan, 2020]. For example, finding a Japanese movie director is simple with a search engine, but gathering specific information like showtimes for films by Japanese directors at local theaters is more challenging. Such tasks typically require humans to manually collect and cross-reference data from various sources, which is time-consuming and may lead to incomplete or approximate results.

Despite the impressive advances in modern ML, machines still struggle to automatically combine and interpret information from different websites for complex tasks, as these require a level of understanding closer to human intelligence [Hogan, 2020]. The root of the issue is that the Web was designed for human consumption, not for machines to extract and process data. To address these challenges, the *Semantic Web* is a vision that considers transforming the Web itself to make its content natively machine-readable. If machines could directly exchange the data on the Web, they could collaborate to present users with more comprehensive and accurate results, automating complex tasks across various domains [Hogan, 2020].

Whether or not the Semantic Web becomes a reality in the future, the field of Semantic Web research has been active for over two decades [Berners-Lee et al., 2001, Hitzler, 2021]. Hitzler [Hitzler, 2021] advocates for not limiting the application of semantic technologies to the World Wide Web, but apply them in general for information integration and management to establish efficient methods and tools for data sharing, discovery, integration, and reuse. This involves following a practical approach and investigating foundations and applications of ontologies, linked data, and knowledge graphs with the W3C standards and recommendations.

The Semantic Web stack, also known as the Semantic Web Layer Cake, represents the architecture of the Semantic Web as a conceptual model, as shown in figure 2.2. It is based on the original idea proposed by Tim Berners-Lee in 2000 [Berners-Lee, 2000], with an updated version [Idehen, 2017] that includes new elements like data lifting and validation.

In the following, we will start to introduce core semantic technologies traversing the Semantic Web stack bottom up starting with the Resource Description Framework (section 2.9), followed by SPARQL (section 2.9). We will then introduce ontologies in section 2.9 & knowledge graphs section 2.9 and conclude the discussion with the contemporary Semantic Web

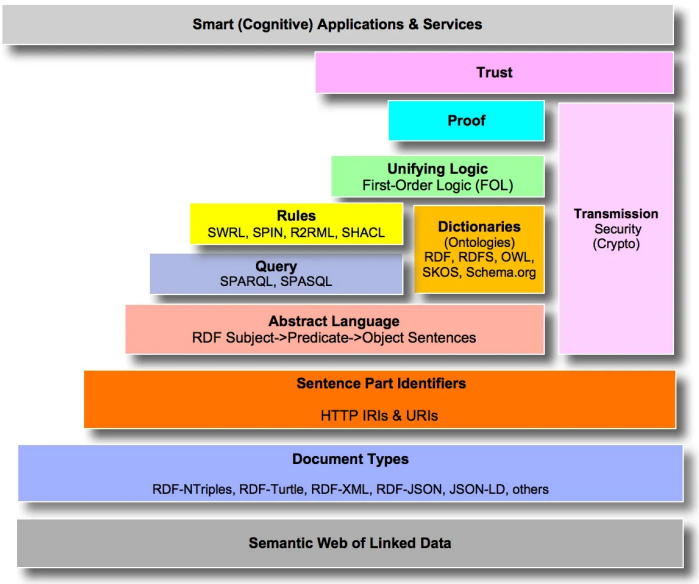


Figure 2.2: Tweaked Semantic Web Technology Layer Cake suggested by [Idehen, 2017], based on the original proposal of Tim Berners-Lee from 2000 [Berners-Lee, 2000].

(section 2.9). These technologies compose the necessary background for a major contribution of this dissertation: The definition of concepts and review of modern techniques for ontology-based data management (see section 5.2).

The Resource Description Framework

"*The Resource Description Framework (RDF) is a standard framework for describing resources*" [Hogan, 2020]. This includes various types, e.g., web content, data attributes, and arbitrary real-world entities (e.g., a city or a person) or abstract concepts (e.g., philosophy or justice). RDF serves as an infrastructure to encode and share (meta)data that captures the structure, relationships, and semantics of data, independent of any specific domain. It is designed to be universally applicable across various fields, focusing on fundamental concepts like entities, attributes, and relationships. It provides a foundation for data organization and integration, facilitating efficient communication and interoperability across different contexts [Theissen-Lipp, 2024]. The framework allows data from different sources to be published in a machine-readable format, reducing the need for format conversions. As such, ideally it addresses the problem of having to integrate heterogeneous data formats by selecting a uniform standard format for publishing and exchanging data on the Web [Hogan, 2020]. RDF was first recognized as a W3C Recommendation in early 2000 [Lassila and Swick, 1999], with a revised version released later in 2004 [Manola et al., 2004]. More recently, in 2014, the RDF 1.1 standard was officially published [Schreiber and Raimond, 2014, Hogan, 2020], introducing minor revisions to the 2004 version, standardizing new syntaxes, and adding some abstract concepts.

RDF uses **Internationalized Resource Identifiers** (IRI) to avoid ambiguity for a specific resource. For example, the word "right" can be interpreted in many ways: as a direction, as

describing correctness, as a mathematical property, or as moral or legal entitlement among others. Hence, pure strings cannot serve as unique identifiers and for this reason the native global naming scheme of the Web can be consulted. The Uniform Resource Locator (URL) refers to an address that specifies a path to a particular file on a server. These can be accessed via the Hypertext Transfer Protocol, the core protocol of the Web standardized by the Internet Engineering Task Force and the W3C [Thomson and Benfield, 2022]. Hence, a generalization for URLs is proposed, with the idea that IRIs serve as unique identifiers for generic resources [Hogan, 2020]. While RDF was originally based on Universal Resource Identifiers (URI) [Berners-Lee et al., 2005], the use of IRIs was introduced in RDF 1.1.

A **literal** in RDF is a syntactic form representing a concrete value, such as a string, number, boolean, or date. Literals are not resources; they do not denote entities in the world and therefore cannot appear as subjects or predicates in an RDF triple. A literal is typically written together with a data type IRI (e.g., `8sd:string`, `8sd:integer`) or, in the case of natural language text, a language tag. Examples are shown below:

```
"Berlin"^^xsd:string
42^^xsd:integer
"2025-12-08"^^xsd:date
"Sun"@en
```

Listing 2.2: Examples of RDF literals

Triples are the core structure of RDF, which represent statements, or facts and are structured as: a subject, a predicate (relation), and an object [Hogan, 2020]. To illustrate how RDF works, consider the example "Earth orbits the Sun." In this case, the subject is "Earth," the verb is "orbits," and the object is "Sun." RDF encodes "Earth" and "Sun" as resources, and "orbits" as the relationship connecting them. By providing a standardized method to define these elements, RDF enables systems to consistently represent and interpret the meaning of these entities and their interactions. Such triples naturally give rise to a graph structure, where statement represents a relationship between two entities, where the subject and object are nodes, and the predicate (or verb) is the edge connecting them. If additional triples are introduced (e.g., "Sun is a star"), the graph grows, with new nodes (like "star") and new edges connecting those nodes.

WikiData[3] is a free, collaborative knowledge base closely linked to Wikipedia, providing structured data to support the representation of knowledge across various domains. It uses RDF principles to represent entities and relationships and thus standardized IRIs uniquely identify resources. For example, *https://www.wikidata.org/wiki/Q2* represents Earth, while *https://www.wikidata.org/wiki/Q525* represents the Sun. In addition to identifying resources, IRIs are also used to define the types of relationships between them. For example, the relationship "orbits", that connects the Earth to the Sun, could be represented as *https://www.wikidata.org/wiki/Property:P397*.

Such **RDF graphs** are particularly useful for representing complex, interconnected data because they allow relationships between elements to be defined explicitly [Hogan, 2020]. The structure of these graphs can handle data from various domains and is inherently flexible,

[3] https://www.wikidata.org/wiki/Wikidata:Main_Page, accessed on 02.05.2026

allowing for the integration of data from different sources without rigid schema constraints. RDF graphs are typically represented using directed labeled graphs, meaning that the edges have both a direction (from subject to object) and a label (the predicate), which makes them highly expressive and suited for representing the semantics of data.

Prefixes prevent RDF graphs from becoming to long and redundant [Hogan, 2020]. The definition of a prefix shortens a URL without loosing its uniqueness. By declaring a prefix like *wd:* and associating it with the namespace *https://www.wikidata.org/entity/*, the full URL *https://www.wikidata.org/entity/Q243* can be shortened to *wd:Q243*. The most common prefix is *rdf:* which stands for *http://www.w3.org/1999/02/22-rdf-syntax-ns#* and refers to the RDF namespace. While this prefixed form is much more concise, it still retains its connection to the original IRI during interpretation.

Namespaces are the shared part of a URI [Hogan, 2020]. For instance, all URIs from Wikidata, like *https://www.wikidata.org/entity/Q243*, share the same starting substring, which identifies the authority (*https://www.wikidata.org/entity/*). Within a single namespace, each entity must have a unique path to ensure that no two distinct entities share the same URI, as this would cause ambiguity and result in conflicts in identifying resources.

RDF Schema (RDFS) is a lightweight extension of RDF that adds a simple vocabulary to define relationships between classes and properties in order to allow inference based on these relationships [Hogan, 2020]. For example, it introduces the *rdfs:subClassOf* property to indicate subclass relationships. If *wd:Planet* (wd:Q634) is a subclass (wd:P279) of *wd:planetary-mass object* (wd:Q400144), any instance of *wd:Planet*, like *wd:Earth*, is also an instance of *wd:planetary-mass object*. Similarly, properties can have hierarchies using *rdfs:subPropertyOf*. RDFS also provides mechanisms to constrain how properties are used, specifically with *rdfs:domain* and *rdfs:range*. For instance, the property *wd:orbits* could be defined with a domain of *wd:Planet* and a range of *wd:Star*, ensuring it can only relate planets to stars in a particular RDF graph. In this case, the statement *wd:Earth :orbits wd:Sun* would imply that Earth is of type *wd:Planet* and *wd:Sun* is of type *wd:Star*. Given the statement *wd:Earth wd:orbits wd:Sun*, we can deduce that Earth is a planet and the Sun is a star, even if that wasn't explicitly stated, because of the domain and range definitions.

Properties and **classes** help to formally organize entities in RDF. Properties define the relationships, while classes group resources conceptually. All resources belonging to the same class share a conceptual classification, but this doesn't necessarily mean they share properties. A resource that belongs to a class is called an instance of that class, and the class of the instance is known as its type. The property *rdf:type* is used to assign a resource to a class. Each relation is defined by a property type, and just like resources, properties are identified by IRIs. The class *rdf:Property* indicates a resource is a property in RDF. For example, if we take "Earth orbits the Sun" as our base, we could define the RDF graph as follows: "Earth" and "Sun" are both resources, and "orbits" is a property that links them. "Earth" is an instance of the class "Planet" (*https://www.wikidata.org/wiki/Q128207*), and "Sun" is an instance of the class "Star" (https://www.wikidata.org/wiki/Q5864). The *rdf:type* property would connect "Earth" to the class "Planet" and "Sun" to the class "Star". The relation "orbits" would be an instance of *rdf:Property*.

```
@prefix wd: <https://www.wikidata.org/entity/> .
@prefix wdt: <https://www.wikidata.org/prop/direct/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .

# Define Earth and Sun as resources
wd:Q2 rdf:type wd:Q128207 ;  # Earth is a Planet
wdt:P397 wd:Q525 .     # Earth orbits the Sun

wd:Q525 rdf:type wd:Q5864 ;  # Sun is a Star
rdfs:label "Sun"@en .

# Defining classes and subclass relations
wd:Q128207 rdfs:subClassOf wd:Q128207 ;    # Planet is a subclass of
    planet
rdfs:label "Planet"@en .

wd:Q5864 rdfs:subClassOf wd:Q523 ;  # Star is a subclass of celestial
    body
rdfs:label "Star"@en .

# Property definitions
wdt:P397 rdf:type rdf:Property ;  # Define the 'orbits' property
rdfs:domain wd:Q128207 ;  # orbits is used for planets (domain)
rdfs:range wd:Q5864 .      # orbits connects planets to stars (range)

# The following triple could be infered from the above
wd:Q525 wdt:P31 wd:Q523 .    # Sun is a celestial body
```

Listing 2.3: Example RDF graph serialized in Turtle format, illustrating key concepts in the context of Earth orbiting the Sun via the WikiData free and open knowledge base

Many of the classes and properties are defined using namespaces from established ontologies (see section 2.9).

There are various formats for **RDF serialization** to save and read RDF graphs. XML is a markup language introduced as a W3C recommendation in 1998 as a file format for sharing structured data across the web [Bray et al., 1998] and initially, XML was the dominant format on the Semantic Web. In the year 2000, Decker et al. [2000] predict that XML will turn out to be ineffective in the long run and indeed over the years we observe an evolution of formats leading to JSON and JSON-LD [Sporny et al., 2019], the N-Triples syntax [Carothers and Seaborne, 2014], and the Terse RDF Triple Language (Turtle) [Beckett et al., 2014]. These formats excel in their simplicity, interoperability and human-readability, so that XML is not as commonly used anymore.

Listing 2.3 illustrates the previously mentioned key concepts such as resources, triples, prefixes, subclass relationships (rdfs:subClassOf), property definitions, and domain-range constraints in the context of Earth orbiting the Sun. Turtle aims at human-readability and utilizes "syntactic sugar" to reduce explicit redundancy in triples. In Turtle, triples that share the same subject can be grouped so that the subject IRI is only defined once. Using semicolons, the following triples are defined using the previous subject implicitly and a single dot "." ends such sentences.

Triple stores are specialized databases that can store, manage and operate large RDF graphs. They offer more robust solutions for applications handling large volumes of triples. Key performance metrics for triple stores include data ingestion speed and query response time.

Additionally, triple stores make their data accessible via standardized query languages (see the next section).

Mapping Heterogeneous Data to RDF: R2RML and RML

While RDF provides a uniform data model for publishing and integrating information, most industrial data is not natively available as RDF but stored in relational tables, CSV files, JSON documents, XML, or domain-specific formats. A key step in semantic data management is therefore the declarative specification of how source data elements are transformed into RDF resources, properties, and literals.

The W3C recommendation *R2RML* [Das et al., 2012] introduced a standard mapping language for generating RDF from relational databases by specifying (i) a logical source (table/view), (ii) rules for constructing subjects (e.g., IRIs from identifiers), and (iii) predicate–object assertions derived from columns. *RML* [Dimou et al., 2014] generalizes this principle beyond relational data by supporting heterogeneous sources such as CSV, JSON, and XML through the notion of a logical source combined with a reference formulation (e.g., column names, JSONPath, XPath). Conceptually, an RML mapping defines a set of rules that select records from a source and construct. For each record, a set of RDF triples whose subjects, predicates, and objects may be formed from constants, templates, and references to values in the underlying record.

Mapping languages such as R2RML/RML operationalize the link between raw data and a conceptual vocabulary: they make explicit how data elements correspond to domain entities, attributes, and relations, and they provide an executable specification that can be evaluated for coverage, correctness, and maintainability. In later sections, this role is leveraged to connect semantic enrichment and integration mechanisms to unified, meaning-oriented access across heterogeneous data sources.

RML extends the W3C R2RML mapping model by reusing the `rr:` vocabulary and adding source-independent constructs for non-relational inputs (e.g., CSV/JSON/XML). A core small subset is represented by:

- `rr:TriplesMap` (R2RML): the central mapping unit. Each triples map specifies how to construct RDF triples from a set of source records.
- `rml:logicalSource` (RML): declares the input and record selection, typically via `rml:source` (the dataset), `rml:referenceFormulation` (e.g., `ql:CSV`, `ql:JSONPath`, `ql:XPath`), and optionally `rml:iterator` to select records.
- `rr:subjectMap` (R2RML): defines how subjects (IRIs/blank nodes) are constructed, commonly using `rr:template` (IRI template) or `rr:constant`.
- `rr:predicateObjectMap` (R2RML): groups one or more predicate–object assertions generated per record.
- `rr:predicateMap` / `rr:objectMap` (R2RML): specify predicates and objects; objects may be constants/templates or extracted from the record via `rml:reference` (RML's generalized field/path reference).

- `rr:parentTriplesMap` + `rr:joinCondition` (R2RML): expresses joins between record sets for generating links across entities (foreign-key style linking).

SPARQL

Just as traditional databases require specific query languages, RDF-based Semantic Web technologies also rely on specialized query languages. The query, written in a machine-readable format, is processed by an engine that interprets the query, identifies the required entities and conditions, and matches the query against the data to return the results. An example of a widely-used query language for structured data is SQL, which operates on relational databases.

Similar to SQL, the **SPARQL Protocol And RDF Query Language** (SPARQL) provides a query language specifically for RDF datasets. Originating from SquishQL in 2002 [Miller et al., 2002], SPARQL was standardized by the W3C in 2008, with the current revision, SPARQL 1.1, released in 2013 [Aranda et al., 2013]. SPARQL allows users to query and manipulate RDF graph content stored in databases or accessible via the Web through SPARQL endpoints, typically accessed over HTTP. SPARQL adopts a syntax similar to Turtle and allows clients to retrieve data from RDF triple stores. If a user constructs a valid query, the requested data is retrieved and transmitted to the user. SPARQL 1.1 encompasses a query language, result formats (like JSON or XML), an update language, and more. Queries can range from simple graph pattern matching to more complex operations, similar to SQL queries in relational databases, and include features like unions, filters, aggregation, and nested queries. SPARQL queries are built on triple patterns, and the SPARQL engine returns results that match these patterns. Listing 2.4 shows a sample SPARQL query that retrieves all planets and their respective stars based on the triples defined in listing 2.3.

```
1  PREFIX wd: <https://www.wikidata.org/entity/>
2  PREFIX wdt: <https://www.wikidata.org/prop/direct/>
3  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6  # Select planets and their corresponding stars
7  SELECT ?planet ?star
8  WHERE {
9    ?planet rdf:type wd:Q128207 ;    # Planet type
10   wdt:P397 ?star .          # The 'orbits' relationship
11   ?star rdf:type wd:Q5864 .        # Ensure the star is of type Star
12 }
```

Listing 2.4: Example SPARQL query retrieving all planets that orbit stars, based on the RDF graph data stored in the WikiData knowledge base

However, these queries must be both syntactically and semantically correct. Syntax errors can be identified by comparing the query against the SPARQL specification, while semantic correctness ensures that the query returns meaningful results. If a query is syntactically correct but uses incorrect classes, predicates, or relations that do not match the RDF data's structure, the query may return no results or incomplete data. To retrieve accurate results, the user must therefore be aware of the RDF data's underlying semantic structure, which defines the relationships between nodes (i.e., classes and properties) in the graph. For smaller RDF datasets, this structure might be easy to visualize, but for larger, more complex datasets, it becomes

challenging. Ontologies (see the next section) can help by providing a blueprint for structuring queries based on predefined relationships and classes.

Ontologies and the Web Ontology Language

Formalizing knowledge to allow machines to understand human concepts has been a major goal in computer science,. This requires a shared conceptualization that can be processed by both humans and machines. Ontologies and similar tools are essential for representing knowledge in structured ways [Gruber, 1993].

Concepts and classes refer to abstractions that generalize ideas or objects by capturing similarities and expressing meaning (i.e., a celestial body) [Studer et al., 1998]. A dictionary is a basic form of knowledge storage that lists pairs of labels and descriptions but does not establish relationships between them. Taking on our previous example, the word "right" has 17 different meanings[4] on dictionary.com. Taxonomies extend dictionaries by organizing concepts into hierarchical structures. Ontologies, the most expressive form, not only provide hierarchical and non-hierarchical relationships but also introduce axioms and constraints, making them domain-specific yet powerful for knowledge representation. Gruber Gruber [1993] defined an ontology as a "formal, explicit specification of a shared conceptualization."

Baader [2003] et al. provide a helpful description for understanding ontologies. Most ontologies contain a so-called *Terminological Box* (TBox), which defines the structure and relationships within a domain (e.g., "In our domain/universe of interest, there exists a class for celestial bodies"), and some also include a so-called *Assertional Box* (ABox), which represents factual knowledge (e.g., "The Earth is such a celestial body"). Knowledge graphs (see section 2.9) often contain more factual data than ontologies, though both provide a framework for organizing knowledge and mapping data attributes to a formal conceptualization.

The **Web Ontology Language** (OWL) was introduced in 2001 and adopted as a W3C recommendation in 2004. While RDF and RDFS serve as foundational tools to formalize an ontology by allowing knowledge engineers to define classes and properties and their constraints, OWL extends this by providing additional mechanisms based on description logic to specify more complex relationships and restrictions. OWL2, an update to OWL, was recommended in 2009 and introduced advanced features to the vocabulary [Bao et al., 2012]. OWL includes vocabulary (*owl:Ontology*) to allow for the specification of new ontologies and their metadata, including multilingual labels and descriptions. The *owl:imports* property can be used to reference and incorporate other ontologies. While RDF and RDFS allow for basic class and property definitions, OWL introduces more advanced features such as cardinality constraints, property characteristics (e.g., transitivity, symmetry), and class expressions (e.g., unions and intersections of classes) to enable more precise and expressive data modeling. Thus, OWL supports better reasoning capabilities, allowing machines to infer new knowledge from the relationships and rules defined within the ontology.

[4] https://www.dictionary.com/browse/right, accessed on 02.05.2026

Knowledge Graphs

Recently Knowledge Graphs (KG) have emerged as a powerful tool for enhancing data usability and interpretation [Hogan et al., 2022, Ehrlinger and Wöß, 2016]. They have evolved significantly since their initial mention in academic literature approximately fifty years ago [Schneider, 1973], with their modern usage gaining prominence through announcement of the *Google Knowledge Graph* (GKG) [Speer et al., 2012].

Rooted in principles of Linked Data and Semantic Web standards, KGs present interconnected knowledge on the Web in a machine-readable format by structuring information as nodes (representing entities) and edges (representing relationships) [Hogan et al., 2022]. This approach allows for the modeling of complex semantic relationships such as hierarchies, classifications, and associations, which are fundamental to the Semantic Web paradigm.

A thorough overview by Hogan et al. from 2021 [Hogan et al., 2022] defines a KG as "a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities." A key feature of KGs is that entities are interlinked to one another; the definition of one entity points to another entity. Alternatively, Ehrlinger and Wöß [2016] emphasize the central role of the ontology: "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge." This is because in the literature the terms ontology and KG are often confused. An example is *YAGO* (*Yet Another Great Ontology*), which is – according to its name – an ontology, but is also referred to as KG [Färber and Rettinger, 2015, Paulheim, 2017]. Hence, a KG needs to contain at least a basic ontology providing the metadata/context in order to enable correct interpretation, i.e., the "semantics" or "meaning" of the stored facts. This structure implies that, compared to an ontology, the ABox (assertions about the world) is larger than the TBox (schema or vocabulary) (see section 2.9). Although the definitions are independent of specific Semantic Web technologies, RDF and OWL can and typically do complement each other, e.g., by using OWL as a set of axioms to specify an ontology/schema of types and their relationships, and by then using these in the RDF graph, which in return contains the factual information; together they form a KG that represents a collection of interlinked descriptions of entities. Not every RDF graph is a KG and not every knowledge base is a KG. For instance, a set of statistical data represented in RDF format is not automatically a KG. It is possible to have an expert system without formal structure and semantics, e.g., a Q&A platform that has a collection of data, but uses automated deductive processes such as a set of 'if-then' rules to facilitate the analysis, which do not represent a KG.

KGs can be divided into two main categories: enterprise and open. Enterprise KGs, such as *Google*'s, store information relevant to a company's business operations, while open KGs, like *Wikidata* and *DBpedia*, provide information accessible to all users. With more than 70 billion assertions describing a billion entities [Noy et al., 2019], the GKG is the result of more than a decade of data contribution activity from a diverse set of individuals that powers APIs for *YouTube* and *Google Cloud*. For example, a query for any more or less famous person

produces a small panel[5] in the search results with information.

Wikidata, a collaboratively-edited KG, serves as a structured information source for applications like *Wikipedia*, aiming to curate data in a language-agnostic manner. In 2024, *Wikidata* boasted over 100 million entries, actively edited by a large community of users. *DBpedia*, on the other hand, extracts structured data from Wikipedia articles, adhering to the guidelines of Linked Open Data (see section 2.9), and regularly releases updated snapshots of its knowledge base. Unlike enterprise KGs, which are typically maintained by a single entity, open KGs are continuously expanded through voluntary contributions from a broad user base.

Linked Data and the Semantic Web

Within the Semantic Web, RDF and OWL are the main languages to represent data as a set of **Linked Data** items [Bizer et al., 2011a]. Linked Data refers to a set of best practices for publishing and connecting structured data on the Web, making it machine-readable and interconnected [Bizer et al., 2011b]. Key principles include using unique, persistent URI to identify resources, creating explicit links between resources, and utilizing standard Web protocols like HTTP. These principles enable the development of a decentralized, distributed web of linked data sources, allowing large-scale data integration and reasoning across the Web. In 2006, Tim Berners-Lee introduced the following foundational rules for Linked Data, now known as the Linked Data principles[6]:

1. Use IRIs as identifiers for entities
2. Use HTTP IRIs to enable dereferencing
3. Return information when an IRI is looked up
4. Include links to other objects

When data is shared under an open access license, it is known as Linked Open Data. The fundamental principles of Linked Open Data are also outlined by Tim Berners-Lee in the 5-Star Linked Data scheme, which provides a framework for assessing the quality of Linked Data across five levels, each with its own set of criteria: These principles resemble are strong simi-

1 Star	Publish your data in an arbitrary format but under an open license
2 Stars	Provide structured machine-readable data
3 Stars	Use non-proprietary formats
4 Stars	Use URIs to identify things in the data
5 Stars	Link the data to other data to provide context

larity to the FAIR principles (section 2.4). Both paradigms not only provide tools and concepts for interlinking of data and metadata, but also give guidance on how to implement helpful actions in practice on how to store valuable information to foster understanding by both human and machine. In 2001, Tim Berners-Lee introduced the concept of the Semantic Web as a vision for improving data handling on the World Wide Web [Lassila et al., 2001]. He envisioned it as an evolution of the current web, where data is not only accessible and understandable

[5] https://support.google.com/knowledgepanel/answer/9163198, accessed on 02.05.2026
[6] https://www.w3.org/DesignIssues/LinkedData.html, accessed on 02.05.2026

to humans but also to machines. In this upgraded "Web of Data" [Hogan, 2020], much of the available information would be represented in machine-readable formats, rather than unstructured forms. Just as web pages rely on structured HTML, semantic information can be handled with RDF, OWL, and SPARQL, forming the foundation of the Semantic Web. Prominent Linked Open Data efforts include YAGO [Rebele et al., 2016], DBPedia [Mendes et al., 2012], the Schema.org vocabulary [Guha et al., 2016], and Wikidata [Cantallops et al., 2019].

Additionally, a significant portion of (linked) open data is published through platforms known as **Open Data Portals**, which function as catalogs similar to digital libraries [Neumaier et al., 2017]. In these portals, a dataset comprises a collection of data files available for access or download in multiple formats, such as CSV or PDF. Each dataset is also accompanied by metadata, which provides structured descriptive information about the resources, including details on authorship, provenance, and licensing. Most Open Data Portals are built on existing software frameworks like CKAN[7] or Socrata[8], which offer functionalities such as user interfaces, search capabilities, and APIs. CKAN is the most widely used framework for publishing open data and powers several governmental portals, including data.gov.uk and data.gov (accessed on 02.05.2026).

[7] https://ckan.org/, accessed on 02.05.2026
[8] https://dev.socrata.com/, accessed on 02.05.2026

3

Defining Data Architectures and their Technical Requirements

> "*The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.*"
>
> Edsger Dijkstra

Data architectures form the conceptual and technical backbone of modern data management systems. As organizations increasingly rely on heterogeneous, large-scale, and fast-changing data, architectural choices shape performance, scalability, interoperability, and the feasibility of analytics. This chapter addresses RQ1 by systematically characterizing the design space of contemporary data platforms and situating SEDAR within it. The need for such an analysis arose from SEDAR's development trajectory (chapter 4): although originally conceived as a classical data lake, it gradually incorporated semantic, metadata-driven, and automation-oriented capabilities that no longer aligned with traditional definitions.

To clarify this landscape, we developed the **DATER** framework (**D**ata **A**rchitectures and their **Te**chnical **R**equirements), which formalizes architectural comparison across well-defined dimensions. DATER serves two purposes:

1. offering a structured scheme for comparing heterogeneous architectural paradigms, and
2. enabling a principled assessment of SEDAR's architectural identity and its alignment with modern requirements.

This chapter therefore provides the conceptual foundation for later chapters on semantic data management (chapter 5) and automated machine learning (chapter 6), where these requirements manifest in SEDAR's implementation.

This chapter is largely based on [Hoseini et al., 2026b].

Chapter Outline

This chapter is structured as follows:

- Section 3.1 — Introduction and motivation for analyzing contemporary data architectures, including the connection to RQ1 and the role of SEDAR in motivating the DATER framework.
- Section 3.2 — Terminology and foundational concepts:
- Section 3.3 — Detailed characterization of seven major data architectures (data warehouse, data lake, semantic data lake, lakehouse, data space, data fabric, data mesh), including historical background, defining principles, structural features, and formal requirements.
- Section 3.4 — The DATER evaluation framework:
  - Section 3.4.1 — Derivation of evaluation dimensions for comparing data architectures.
  - Section 3.4.2 — Systematic mapping of real-world platforms against the DATER framework to assess how theoretical models manifest in practice.
- Section 3.5 — Overview of related surveys and research efforts on data architectures, highlighting gaps addressed by DATER.
- Section 3.6 — Concluding discussion of how the chapter addresses RQ1.

3.1 Introduction

Data Architectures play a fundamental role in data management. Given that most organizations possess far more data than any individual can fully grasp, representing data at different levels of abstraction becomes essential for comprehending and processing it. In general, architecture refers to the organized arrangement of components designed to optimize functionality, performance, feasibility, cost, and aesthetics of a system. In information technology, a data architecture describes various aspects of system design for data management in order to provide guidance for the development of new versions or extensions of the system [International, 2017, Muller, 2008]. The common goal is the effective management of data and the various systems or components in which it resides [International, 2017]. At the most detailed level, a data architecture is encapsulated in an organizations data model, which includes for example data names, comprehensive data and metadata definitions, conceptual and logical entities, relationships, and business rules. Data architectures are most effective when they support the entire organization, enabling consistent data standardization and integration across all data-related activities [International, 2017, Gröger, 2021].

Data architecture designs may thus vary for different levels within an organization, for particular domains, or based on its focuses such as infrastructure, applications, or data. To avoid confusion, architectural frameworks help bridge this understanding by providing a structured way to comprehend the various design choices within systems. This chapter aims at giving an overview on the predominant frameworks for scalable data platforms in the modern age. This is important because substantial changes in enterprise data management usually mean a huge organizational effort. Gaining insights into the experiences, strategies, implementations, and deployment processes involved in transitioning to a modern architecture—while identifying the

challenges encountered—helps mitigate costly pitfalls and prevents data mess.

We will give an overview of each individual framework for data architectures including its historical development and then discuss their main differences and similarities. One may view the description of architecture type as a so-called **reference architecture** [Sang et al., 2016, 2017]. These serve as blueprints for developing concrete solutions based on a architecture, offering standardized guidelines for the design. They include a shared vocabulary, best practices, architectural components, and design principles.

As described in section 1.1, this research began with the development of a data platform for the research project i²DACH to support the management of heterogeneous process data and ML methods for analyzing this data. This resulted in the development of the ***SE**mantic **DA**ta **R**eservoir* [Hoseini et al., 2023] (SEDAR) whose architecture is described in detail in chapter 4. The discussion in this chapter will help assign the architecture of SEDAR to a particular framework.

Choosing the appropriate data architecture is critical, as substantial changes in enterprise data management typically require significant organizational effort. Gaining insights into the experiences, strategies, implementations, and deployment processes involved in transitioning to a modern architecture helps mitigate costly pitfalls and prevents data mess. We focus on the technical characteristics for particular data architectures including their historical development and then discuss main differences and similarities. Our descriptions of the different architectures are intended to support the development of reference architectures [Sheth and Larson, 1990, Sang et al., 2016, 2017]. These reference architectures serve as blueprints for designing concrete solutions based on specific architecture types, offering standardized design guidelines [Van den Hoven, 2003] and fostering a shared vocabulary.

Furthermore, we propose a comprehensive framework for evaluating and classifying various data architectures referred to as Data **A**rchitectures and their **Te**chnical **R**equirements (DATER). DATER provides valuable insights into the evolving landscape of data systems, helping both researchers and practitioners make informed decisions about which architectural models best suit their specific needs and goals. While many real-world systems do not fully align with theoretical models, DATER enables a precise identification of which architectural components or capabilities differ. This helps bridge the gap between abstract design principles and practical implementations, guiding more effective system development and evaluation.

Scope The data architectures and frameworks we discuss reflect the evolving landscape of data management, selected for their prominence in current discourse and their ability to meet emerging demands in data-intensive environments.

A wide range of additional architectures exists, such as data hubs, platforms, vaults, grids, multi-cloud, hybrid and federated architectures as well as lambda & kappa, knowledge graph-based or event-driven data architectures, [Linstedt and Olschimke, 2015, Kiran et al., 2015, Lin, 2017, Hoseini et al., 2023, Plattform Industrie 4.0, 2019, Venugopal et al., 2006]. Here, we focus on those that have gained substantial traction in both academic and industry discussions. As Ataei and Litchfield [2022] point out, in many cases, many of these additional terms can be

traced back to one of the core architectures discussed here. For example, data vaults can be considered as an extension of data warehouses [Linstedt and Olschimke, 2015, Inmon and Linstedt, 2014]. Furthermore, the lambda and kappa architectures address how to *process* stream data on a large scale, but do not state how to manage or organize data repositories.

Research Methods To collect relevant scientific articles, we made use of Google Scholar primarily. To get an overview, for each architecture we selected roughly 20 well-cited journal papers, sorted them by relevance and began reviewing the content. We made heavy use of the citation snowballing method Badampudi et al. [2015], i.e., extending the results by searching through the cited references in the initial keyword-based findings as well as exploiting the 'Cited By' functionality in Google Scholar. The systems listed in the evaluation were found using Google Search, Perplexity, and GitHub. We derive the evaluation criteria in section 3.4.

Contributions In summary, this chapter makes the following contributions:

1. We first define our terminology to clarify key terms such as data ecosystem, data architecture, and data platform (section 3.2).
2. The core of the DATER framework is the characterization, definition, and identification of technical requirements for six data architectures (section 3.3).
3. From this, we extract nine evaluation dimensions to assess data architectures. Moreover, a comparative evaluation of real-world systems is conducted by mapping them against the DATER framework, providing insights into their alignment with various architectural models (section 3.4).

3.2 Terminology and Requirements for Data Architectures

Terms such as architecture, framework, ecosystem, platform and infrastructure are strongly related and often used interchangeably, causing confusion and misinterpretation [Giebler et al., 2021]. For example, Geisler et al. [2021] give one particular definition for the term data ecosystem, but point to *Ontario* [Endris et al., 2019] and *Constance* [Hai et al., 2016] as concrete implementations, which are titled as (semantic) data lake by the authors. While DMBOK [International, 2017] distinguishes between data architecture, framework, and strategy, their descriptions occasionally overlap, reflecting the inherent ambiguity of terminology in this domain. Figure 3.1 illustrates the introduced terminology in a hierarchy of data strategy, ecosystems, governance frameworks, architectures, and technical components. Each concentric ring moves from abstract, organizational-level concerns (such as strategy and governance) toward concrete, technical implementations, e.g., data architectures and individual components for data processing. The inner layers in the core of the figure highlight architectural aspects, grouped first by the selected architectures in this article, then by central technical functionality, and finally with a selection of concrete technologies for individual components.

The concepts data strategy, data ecosystems, data governance framework and data architecture describe different perspectives of data management. In the following, we aim to give

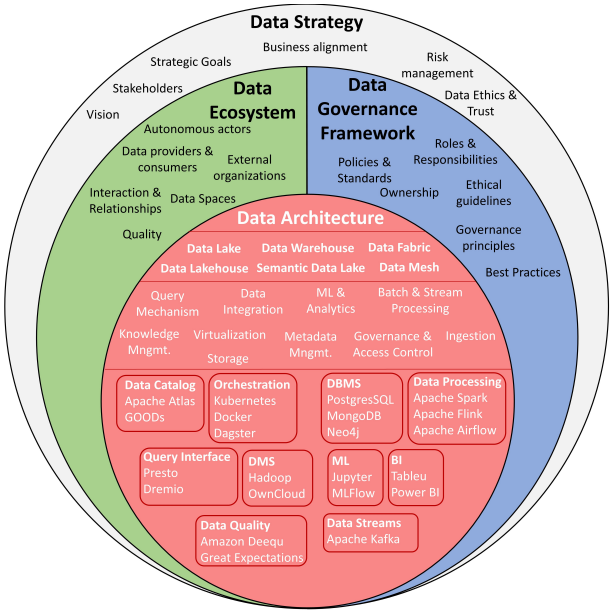


Figure 3.1: Overview of terminology, abstractions and components for data architectures.

precise definitions, while being aware that others may use the terms differently depending on the context:

Data strategy refers to a high-level plan for how an organization will collect, manage, and use data to achieve goals, which must be directly aligned with the overall business strategy [International, 2017].

Data ecosystem, as defined by Oliveira and Lóscio [2018], refers to "*set of networks composed of autonomous actors, which consume, produce, or provide data or other related resources*". It describes the information management landscape of an organization on a conceptual level, i.e., how information is used or produced in business processes as well as how it is managed in information systems. It might include relationships to external organizations, e.g., for information exchange along the supply chain.

A **data governance framework** defines best practices, standardized roles, processes, technologies, and guiding principles for managing data assets across their lifecycle, in alignment with the data strategy of the organization. It defines the governance principles of the data ecosystem of an organization.

Data architecture refers to the technical design of multiple data-related components and their interactions. It should be developed according to the guidelines and goals defined in the data strategy and governance framework.

Data platform, in this article, refers to a concrete implementation of a data architecture.

To clearly distinguish each term, we address potential points of confusion by discussing areas where their meanings may overlap.

Data ecosystems are usually composed of multiple data architectures, i.e., an organization might have data warehouses as well as data lakes. The abstraction at data ecosystem level should not only focus on technical aspects and tools, it should also include people and poli-

cies. In this sense, the term ecosystem emphasizes the sociotechnical dimension, focusing on actors, governance structures, interoperability standards, and shared infrastructure.

In this sense, we illustrated data governance frameworks as complementary to data ecosystems. Data ecosystems rather focus on actors, components, and their relationships, whereas a data governance framework aims at policies, standards, responsibilities, and guidelines.

The core of Figure 3.1 illustrates common functions and components of data architectures. For example, we view database management systems, data management systems, and data catalogs as individual components of data architectures. We distinguish between *Database Management Systems* (DBMSs) and the broader category of *Data Management Systems* (DMSs). A DBMS is a software system that supports the definition, querying, and manipulation of structured data, typically offering declarative query languages (e.g., SQL), transaction management, and consistency guarantees [Date, 1977]. DBMSs, but also DMSs, are central components of many data architectures and are usually tailored to a specific data model, which defines their query capabilities.

While traditional relational DBMSs remain dominant, recent years have seen the rise of specialized systems:

- NoSQL DBMSs, categorized into document-oriented (e.g., MongoDB), column-oriented (e.g., Cassandra, HBase), graph-oriented (e.g., Neo4j), and key–value stores Fathy et al. [2019],
- Vector DBMSs (e.g., QDrant) for similarity search and AI workloads,
- Time-series DBMSs (e.g., InfluxDB) for stream and sensor data, and
- Search-oriented DBMSs (e.g., Elasticsearch) for full-text indexing and distributed search.

In contrast, a *Data Management System* (DMS) provides basic mechanisms for storing, accessing, and organizing data, often in the form of files or objects, but lacks advanced DBMS features such as declarative query languages, fine-grained transaction support, or schema enforcement. DMSs prioritize scalability and availability over strict structure, and typically provide only basic access methods such as file download. An example is Apache Hadoop. Another important component of a data architecture is a *Metadata Management System* or *Data Catalog*, which is a system used to manage and organize metadata. Data catalogs provide capabilities for tracking data lineage, enabling data search and discovery, and ensuring that data is well-documented and accessible for various users. They play a crucial role in data governance and help organizations maintain compliance, ensure data quality, and enhance the overall usability of data.

3.3 Data Architectures

With the terminology and general requirements in place, we now introduce and analyze seven prominent data architectures: data warehouse (DW), data lake (DL), semantic data lake (SDL), data lakehouse (DLH), data fabric (DF), and data mesh (DM). Each architecture is presented with a brief discussion of its historical motivation, followed by a definition, characteristic features, formal requirements, and an architecture diagram.

Figure 3.2 illustrates the general historical development and conceptual relationships among

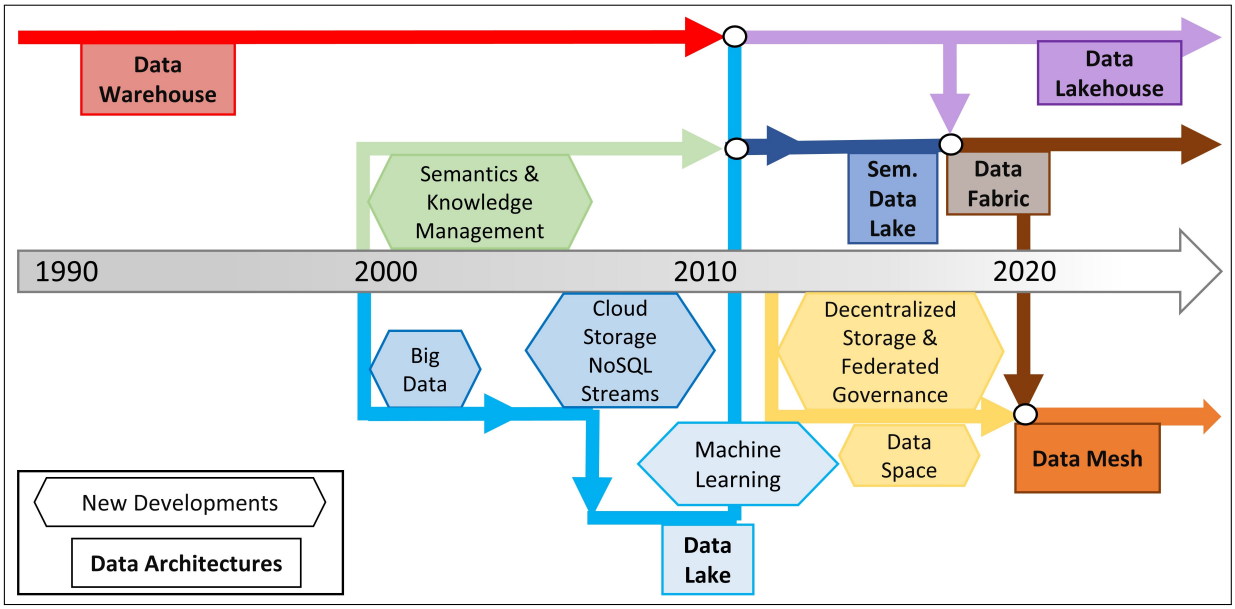


Figure 3.2: Evolution of the various discussed data architectures

these architectures. The evolution begins with data warehouses, which emerged in the 1990s to support analytical reporting and business intelligence through centralized, materialized, and integrated repositories [Inmon, 2005]. An important precursor was the federated database system of the 1980s, which offered virtual integration of heterogeneous databases without physically centralizing the data [Sheth and Larson, 1990]. The Semantic Web, introduced around 2000 [Decker et al., 2000], laid the groundwork for knowledge representation and semantic integration. Around the same time, the term *big data* was introduced [Laney et al., 2001], emphasizing challenges in volume, velocity, and variety.

The 2000s and early 2010s saw a range of technological shifts: the rise of *NoSQL* database systems to address schema flexibility and scalability [Moniruzzaman and Hossain, 2013], the emergence of cloud storage platforms such as Amazon S3 [Armbrust et al., 2010], and the increasing relevance of data stream processing for real-time analytics [Golab and Ozsu, 2022]. The machine learning (ML) boom, accelerated by advances in deep learning from 2012 onward [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, He et al., 2016], further increased demand for architectures capable of handling heterogeneous, high-volume, and rapidly changing data. In response, data lakes emerged around 2010-2015 [Dixon, 2010], offering scalable storage with schema-on-read and ELT paradigms. These were later extended into semantic data lakes, which incorporate ontologies and metadata for improved data understanding and integration [Hoseini et al., 2024d].

As data became increasingly distributed across departments, organizations, and jurisdictions, new architectural paradigms emerged to address governance, sovereignty, and scalability. Data mesh introduces domain ownership and a “data-as-a-product” philosophy to promote scalability and accountability within organizations [Dehghani, 2022]. Around 2019, data fabric was proposed [Zaidi et al., 2019] as a metadata-driven integration layer that leverages AI for

governance and enables self-service access across heterogeneous environments. More recently, the data lakehouse concept has gained traction [Armbrust et al., 2021], combining the reliability and ACID guarantees of data warehouses with the flexibility of data lakes to support unified batch and real-time analytics.

Overall, the figure highlights a clear evolution from centralized and structured systems toward decentralized, federated, and semantically enriched architectures. This trend reflects the increasing heterogeneity of data sources and the distributed nature of modern enterprises. Nevertheless, data warehouses, lakes, and lakehouses remain central in practice due to their maturity and strong theoretical foundations; especially in scenarios where decentralization is not a primary requirement and user groups are relatively homogeneous [Dehghani, 2022].

3.3.1 Data Warehouse

In the 1990s, data warehouses [Inmon, 2005] emerged as a standard solution for enterprise-wide analytical data management. They enabled organizations to run overnight ETL processes (Extract-Transform-Load) that extracted data from heterogeneous sources, transformed it to meet analytical requirements, and loaded it into a centralized, materialized repository. This architecture supported business intelligence by providing a consistent and integrated view of enterprise data.

Traditionally based on relational database technology, data warehouses are the most established form of analytical data architecture [Inmon, 2005] and remain widely used in industry. They have evolved to support cloud-native implementations that offer greater scalability and lower operational costs. For instance, *Apache Hive* [Huai et al., 2014] introduced data warehouse functionality for big data by enabling SQL queries on top of *Hadoop*, while *Snowflake* [Hentschel et al., 2025] exemplifies a commercial, cloud-native data warehouse. Despite these innovations, data warehouses primarily support predefined analytical queries for reporting purposes and are less suited for exploratory or unstructured analytics such as text or web mining [Chaudhuri and Dayal, 1997, Bose, 2009].

Characteristics

Data warehouses are designed to manage structured data and often rely on multidimensional data models for complex analytical tasks. They typically ensure atomicity, consistency, isolation, and durability (ACID) properties and offer advanced capabilities such as time travel, data governance, and zero-copy cloning [Armbrust et al., 2021].

For the definition we rely on Inmon [2005]:

Definition 1. [**Data Warehouse**] "*A data warehouse is a subject oriented, integrated, non-volatile, and time variant"* data architecture *"in support of management's decisions.*" [Inmon, 2005]

Subject-oriented refers to the purposeful organization of data to support business analysis and decision-making. *Integration* is the central aim: although data is collected from diverse systems, the warehouse presents a unified view to the user. *Non-volatility* indicates that data is

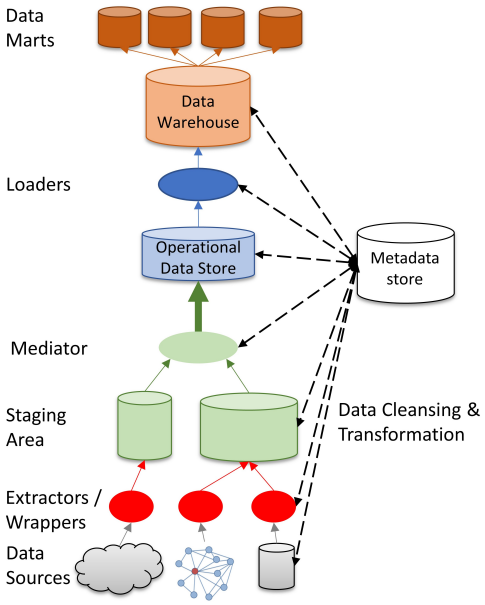


Figure 3.3: Data warehouse architecture based on [Jarke et al., 1999, 2003]

not regularly updated by operational systems but remains stable to support long-term analysis. *Time-variance* reflects the inclusion of historical data, enabling trend analysis and retrospective insights.

In practice, the term “data warehouse” may also refer to centralized repositories that include little or no historical data. However, the core idea remains that it provides business users and decision-makers with consistent, reliable access to analytical data [Harby and Zulkernine, 2024].

The architecture, illustrated in Figure 3.3 [Jarke et al., 1999, Kimball and Ross, 2013], revolves around ETL processes that consolidate data from various sources. While the core storage is generally relational, the staging area may involve other systems, including NoSQL stores [Davoudian et al., 2018]. To support performance and modular access, data marts are employed as thematic subsets of the warehouse, optimized for *Online Analytical Processing* (OLAP) [Chaudhuri and Dayal, 1997]. Data sources are independent systems from which data is extracted. The staging area serves as a buffer for cleansing and transforming data before integration. A mediator integrates this harmonized data into a consistent, unified store. The operational data store holds historical data independently of specific use cases. The data warehouse acts as the central, structured repository optimized for analysis. Data marts are thematic subsets designed to support modular and secure data access. Finally, the metadata system manages schema and lineage information, enabling effective data governance and understanding.

Requirements

The following technical requirements capture the essential features of a data warehouse (Inmon [2005], Chaudhuri and Dayal [1997], Jarke et al. [2003]):

DW1 — Schema-on-write: data must conform to a predefined schema before it is written and stored. Hence, an ETL process consolidating data from heterogeneous sources is required to transform the data into the desired target schema.

DW2 — Structured data model: data is organized into a structured schema following a relational or multidimensional model.

DW3 — Consistency guarantees: provide means for checking and enforcing the consistency of structure and content across data collections.

DW4 — Optimize for read-heavy workloads: optimize query performance, making it ideal for analytical and reporting tasks.

DW5 — Time-variant data storage: include historical data, supporting trends and time-based analysis by retaining snapshots of data over time.

DW6 — Query language: offer a declarative, structured query language.

DW7 — OLAP: enable Online Analytical Processing (OLAP) by supporting multidimensional queries for business intelligence.

3.3.2 Data Lake

Since their emergence, numerous proposals have addressed data lake architectures, in particular, challenges, concepts, components, and implementations [Quix and Hai, 2019, Sawadogo and Darmont, 2021, Scholly et al., 2021, Hai et al., 2016, Zhao et al., 2021a, Farid et al., 2016, Galhotra and Khurana, 2020, Dibowski et al., 2020, Eichler et al., 2021]. In addition, major IT companies have introduced commercial tools for data lakes [Nambiar and Mundra, 2022, Ramakrishnan et al., 2017, Halevy et al., 2016]. In the following, we focus on widely recognized concepts and representative architectural patterns discussed in current academic literature or industrial implementations [Azzabi et al., 2024, Hai et al., 2023, Nargesian et al., 2019].

In 2010, James Dixon [Dixon, 2010] introduced the term data lake as a solution for managing raw data from a single source while accommodating diverse user needs, because "*data is of a scale or daily volume such that it won't fit technically and/or economically into a relational DBMS"* [Dixon, 2010]. This concept stood in stark contrast to the so-far dominant data warehouse systems. By preserving data in its raw format, data lakes bypass or delay the costly preprocessing steps. Dixon motivates the choice for the word lake with an analogy: "*If you think of a datamart as a store of bottled water – cleansed and packaged and structured for easy consumption – the data lake is a large body of water in a more natural state.* " [Dixon, 2010].

In 2014, Gartner [Gartner, 2014] raised several criticisms of data lakes, warning that the indiscriminate ingestion of disparate data could lead to a so-called data swamp, making the data unusable without proper metadata management and governance [Quix and Hai, 2019]. Further, concerns included insufficient attention to data security and privacy oversight. In response to these critiques, Dixon revisited the data lake concept [Dixon, 2014].

With Hadoop at the core, a key requirement was that a data lake should offer low-cost, easily accessible storage. This supports a **schema-on-read** approach, meaning the metadata

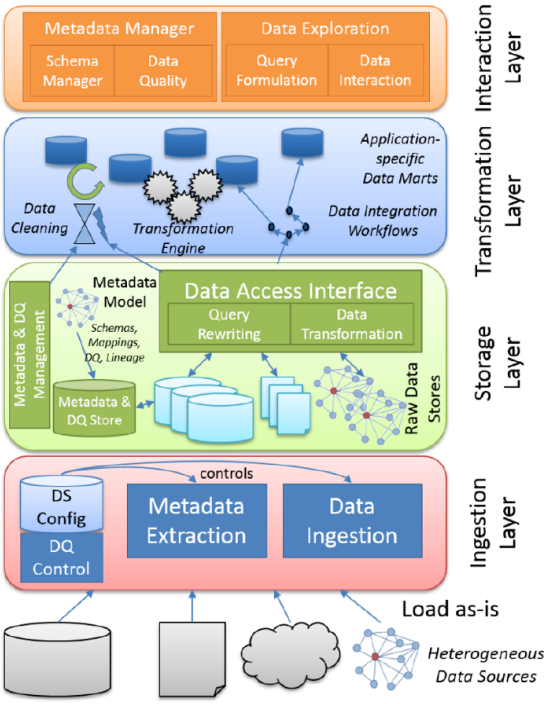


Figure 3.4: Four-layered data lake architecture [Jarke and Quix, 2017]

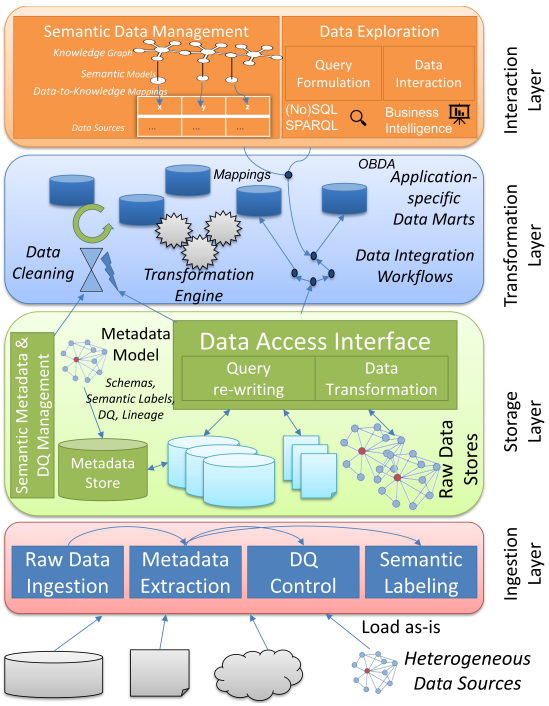


Figure 3.5: Semantic data lake architecture [Hoseini et al., 2024d]

can be populated over time and transformations to apply the schema are only performed once required. This approach allows a data lake to automatically extract metadata from raw data and identify patterns, with users contributing additional descriptive information, such as domain-specific knowledge and attribute linkages. Hence, the idea is that the interaction between users and the data lake continually enhances (meta)data quality and value.

Characteristics

The classical ETL approach was too limited to handle the requirements at the time, because transforming data to fixed schemas and rigid data models leads to limited flexibility in data management. Additionally, some information might be lost during the process, as it might not fit into the target schema and scaling often required a substantial financial investment, both in terms of initial hard- & software costs and ongoing maintenance [Azzabi et al., 2024].

In contrast, data lakes load **raw data** from multiple sources into a unified management system with a heterogeneous storage layer. They provide flexible data access, automated and efficient data preparation workflows, and a series of data pipelines for a wide range of analytical applications of which end-to-end machine learning workflows become increasingly important [Hai et al., 2023, Hoseini et al., 2024c]. Data is transformed only if required for a specific application or query (schema-on-read). Thus, data lakes apply an Extract-Load-Transform (**ELT**) approach. Query processing needs to deal with heterogeneous data sources in an on-the-fly manner [Hai et al., 2023], as data is not transformed into a uniform format before

it is loaded.

Further, data lakes are defined by their ability to scale to massive volumes, all while offering **low-cost storage** solutions typically based on Hadoop, enabling organizations to store petabytes of data [Azzabi et al., 2024].

Apache Spark is a system frequently used for the implementation because it can process queries over heterogeneous data sources. For example *Constance* [Hai et al., 2016] implements a polystore by storing diverse raw data according to its original format: relational (e.g., *MySQL*), document-based (e.g., *MongoDB*), and graph databases (e.g., *Neo4j*).

A key feature is the use of metadata to manage, organize, and analyze the stored data, because otherwise the data is hardly usable as their structure and semantics are not known. Hence, **metadata extraction** or **metadata enrichment**, the process of automatically discovering metadata information of a dataset, is essential for accessing datasets in a later phase. This includes all types of metadata, the more, the better: structural, but also descriptive information as well as relationships to other datasets. Metadata models answer the question of which information to collect and how to structure and organize it and there numerous proposals in recent years [Hoseini et al., 2024c, Scholly et al., 2021, Quix et al., 2016, Sawadogo et al., 2019, Eichler et al., 2020, Ouellette et al., 2021].

Additionally, data lakes aim to enable data exploration and discovery, allowing analysts to navigate and examine heterogeneous data sources without needing predefined schemas or structures and this is a complex challenge and subject to ongoing research [Paton et al., 2023, Hoseini et al., 2023, Mami et al., 2019a, Endris et al., 2019, Bagozi et al., 2019]. Recently, the increasing need for supporting machine learning processes inside a data lake system is emphasized [Hai et al., 2023, Nargesian et al., 2019].

We adopt our definition for data lakes from [Hoseini et al., 2024d] and added further technical requirements:

Definition 2. [**Data lake**] A data lake is a data architecture that manages heterogeneous data sets in their original structures. It should provide functions to extract data and metadata from heterogeneous sources, manage the data efficiently in heterogeneous storage systems, and query and transform the data in a scalable way. Furthermore, a uniform query processing mechanism is necessary, enabling users to query data seamlessly across diverse formats and sources without requiring extensive pre-processing or schema transformations [Quix and Hai, 2019, Hai et al., 2023].

Requirements

DL1 — Schema-on-read: allow raw data of any format to be stored. Schema definitions are only required on access to the data.

DL2 — Data ingestion at scale: support diverse data ingestion methods, such as batch processing, streaming, and real-time data flow for arbitrary heterogeneous data sources.

DL3 — Scalability and cost-effective storage: handle vast amounts of data with elasticity to expand or contract storage based on needs for cost efficiency.

DL4 — Data governance and metadata management: provide a robust data catalog with automated metadata extraction, searchable indexes, tags, and descriptions for datasets ensuring data quality, lineage for data discovery to locate, understand, and utilize data independently of analysis.

DL5 — Unified query mechanism: enable querying of heterogeneous data sources using a unified query interface allowing users to perform federated queries across structured and unstructured data with on-demand data transformations.

Azzabi et al. [2024] outline how the initial *mono-zone architecture* had notable limitations and hence data pond and zone-based architectures were introduced. Zone-based architectures include transient landing, raw, trusted, and refined zones, along with a sandbox zone for exploratory analysis. Each zone has a specific function based on its refinement level, from raw data ingestion and preliminary processing to serving compliant, quality-checked data to end-users [Sharma, 2018, Patel et al., 2017, Ravat and Zhao, 2019, Madsen, 2015, Zikopoulos et al., 2015, Gorelik, 2016]. A *four-layered data lake architecture* was proposed in [Jarke and Quix, 2017] and is shown in figure 3.4. It resembles some ideas of zone-based architectures, such as a raw data 'zone' at the storage layer, and refined data in data marts at the transformation layer. In addition, it emphasizes the role of metadata and quality management on each of the layers.

The *Ingestion Layer* loads data "as-is" into the data lake while maintaining its original format and structure thereby facilitates the acquisition of raw, heterogeneous data from various data sources, including databases, files, and streaming platforms. It is responsible for metadata extraction to provide insights into the structure, quality, and other properties of the incoming data.

The *Storage Layer* provides a centralized repository for storing raw data and metadata retaining the original format of ingested data. It manages metadata and quality information, enabling efficient querying and processing and ensures that raw data remains immutable, supporting traceability and reproducibility.

The *Transformation Layer* is responsible for the integration and preparation of data transforming raw data into refined datasets to meet analytical requirements, it facilitates query rewriting and data transformation tasks, such as data cleaning, enabling seamless data retrieval and manipulation.

The *Interaction Layer* focuses on enhancing the accessibility and usability for end-users acting as the user-facing interface. It comes with some form of metadata manager that handles schema mapping, data quality assessment, and metadata management as well as data exploration to enables users to explore datasets, query the data lake, and interact with application-specific workflows.

3.3.3 Semantic Data Lake

Various strategies for the use of semantic data management have been introduced [Dibowski et al., 2020, Paulus et al., 2021a, Hoseini et al., 2024d], which underscore the necessity of

integrating big data with semantic web technologies into *semantic data lakes*. A semantic data lake facilitates the derivation semantic models from raw datasets, thus providing a more conceptual data description that includes concepts and their interrelations, and linking data sets to their relevant concepts [Dibowski et al., 2020]. Recently, with the growing demand for knowledge management, knowledge graphs have also been integrated into data lake architectures [Dibowski and Schmid, 2021, Beheshti et al., 2018].

At the company *Bosch*, the potential of semantic technologies for data management has already been exploited at scale in production [Kalayci et al., 2020]. Dibowski and Schmid [2021], Dibowski et al. [2020] describe a metadata model represented as an ontology based on W3C recommendations for the data catalog management and provenance control of a semantic data lake. Ingested data assets are **enriched with semantics** by aligning, annotating, and enriching the input data with semantics. The development of knowledge graphs required an immense initial effort by the company but promised a long-term solution for a series of sophisticated challenges. In addition, they utilize a new concept called ontology-driven, self-adaptive frontends, in which user interfaces for data catalogs can dynamically render information from a knowledge graph. A similar concept was presented in [Calvanese et al., 2021]. This is useful because changes in the underlying ontology do not require any changes at the frontend. The *ESKAPE* system from 2017 [Pomp et al., 2017] can be viewed as a very first prototype in this direction and the term was coined by Pomp et al. [2018] and Mami et al. [2018] for the first time in 2018.

Characteristics

We presented a thorough review of the field in [Hoseini et al., 2024d] from which we derive the following definition and requirements:

Definition 3. [**Semantic data lake**] Semantic data lakes are a specific form of traditional data lakes that extend the capabilities through a *semantic layer* that enriches and connects the stored data *semantically*. The *semantic layer* equips data sets in the lake with connections from the data set's metadata to conceptual/logical models, which encapsulate knowledge potentially external to the content of the data, such as domain knowledge.

Requirements

The following, together with the requirements for a traditional data lake (see section 3.3.2), confine the semantic data lake.

SDL1 — Creation of semantic labels: enable the mapping of data attributes and other schema elements to elements of a KG (might be internal or external, domain-specific).

SDL2 — Semantic relationships: represent any form of hierarchical, generic, or pre-defined semantic relationships (semantic connections between data sets, e.g., for provenance or governance).

SDL3 — Metamodel as ontology: the metamodel of the lake itself is modeled as ontology using semantic web technologies.

SDL4 — Semantics-based data access: the ability to query and access data using semantic-based query languages like SPARQL, i.e., abstracting from physical storage and allowing to interact with the data conceptually using techniques like ontology-based data access [Xiao et al., 2018].

In [Hoseini et al., 2024d], we introduce a semantic data lake architecture (see figure 3.5) based on the four-layered architecture. Dibowski and Schmid [2021] describe a semantic data lake as a "*specific form of data lakes in which a semantic layer on top enriches and connects the data semantically*". This definition suggests extending the four-layered architecture by a fifth layer. In our view, proper semantic data management is only possible if it is an integral part of the data architecture. Therefore, incorporating semantics demands adjustment across the entire functionality, starting possibly already at the ingestion, where users (or AI-based technologies) annotate a dataset with semantic labels, before it is fully available inside the lake for accessing. Thus, each layer of the architecture has been extended with enhanced metadata functions with semantic capabilities. For instance, semantic labeling in the ingestion layer assigns semantic labels to metadata elements. Data quality management benefits from an expanded semantic metadata repository managed within the storage layer, which supports functions such as ontology-based data access (OBDA) [Hoseini et al., 2024d], aiding in data usage and interpretation. Consequently, the interaction layer incorporates additional features, such as exploration of the knowledge graph and the semantic model, semantic query formulation using languages like SPARQL, and tools for refining semantic mappings and models.

3.3.4 Data Lakehouse

Although data warehouses and data lakes serve different analytical purposes, organizations frequently require features of both worlds simultaneously [Gröger, 2021]. Especially the necessity of generating business reports and crafting ML models in a single platform gave rise to the *data lakehouse*. Armbrust et al. [2021] recognize that many organizations host an expensive two-tier architecture which is highly complex for users. Data is first ingested into lakes, and then again fed via an ETL process into data warehouses, creating complexity, delays, and new failure modes. Consistency is one of the challenges, i.e., keeping the data lake and warehouse in a consistent state is difficult and costly. Data is stale, new data is loaded to the data warehouse only with a delay due to the ETL process. Finally, both architectures provide limited support for advanced analytics.

The typical data warehouse supports well-designed business intelligence (BI) queries, which are known in advance. The multidimensional data model of data warehouse is optimized to support these queries. However, ad-hoc queries to extract large of amounts of data for a new purpose, e.g., training data for ML models, are less efficient in a data warehouse context. Data lakes, on the other hand, lack the rich and matured features from data warehouses, such as ACID transactions, data versioning and indexing [Armbrust et al., 2021]. In response, Databricks, the company founded by the creators of *Apache Spark*, introduced the lakehouse architecture to combine the best features of both systems. Building on their ex-

pertise with Spark's scalable data processing capabilities, they introduced *Delta Lake* in 2019 [Armbrust et al., 2020] as an open-source storage layer that brings ACID transactions, schema enforcement, and data versioning to data lakes.

Characteristics

Since key characteristics of data warehouses and data lakes often conflict (particularly in terms of data access, data independence, and storage formats) combining them typically requires compromises on both sides. For instance, a data warehouse that supports direct access to storage, as in data lakes, must sacrifice data independence and adopt open file formats. Conversely, a data lake can introduce warehouse-like management features only by restricting users' flexibility in storing and accessing data, enforcing structured protocols for read and write operations [Schneider et al., 2023]. The data lakehouse architecture merges the flexibility of data lakes, which support the large-scale storage of various data formats, with the transactional integrity of data warehouses by incorporating layered components.

Characteristic for a data lakehouse is a **transactional layer**, which ensures data integrity through ACID properties (Atomicity, Consistency, Isolation, Durability). This layer organizes the data and manages metadata, enforcing schema, supporting data versioning, and ensuring data lineage, which enhances overall data quality. Here, technologies like Delta Lake, Apache Hudi, and Iceberg used together with Spark on top of scalable and cloud-native storages like HDFS form the core [Schneider et al., 2024]. Thus, the lakehouse has a **relational format** as its final structure, on top of heterogeneous sources, as most analytical and visualization tools are optimized for relational data, enabling faster analysis. Schneider et al. [2024] conduct a thorough review on the data lakehouse and find four common definitions in contemporary literature and argue that each one of them has certain shortcomings. Hence, they propose the following and derive technical requirements from it. The following are direct quotes from [Schneider et al., 2024]:

Definition 4. [**Data Lakehouse**] *"A data lakehouse is a"* data architecture *"that leverages the same storage type and data format for reporting and OLAP, as for data mining and machine learning, as well as streaming workloads"*.

Requirements

DLH1 — ***"Same storage type and data format****: all (meta)data is stored on a single type of storage (no polystore), using a uniform open data format. Hence, a data lakehouse must have an inherently low heterogeneity in terms of the utilized technologies and data formats.*

DLH2 — ***Relational data collections****: this strong requirement comes from the need to support reporting and OLAP workloads, which require a higher degree of structure than would be provided by other types of abstractions, such as graphs or documents.*

DLH3 — ***Query language****: offer a declarative, structured data query language for the data collections.*

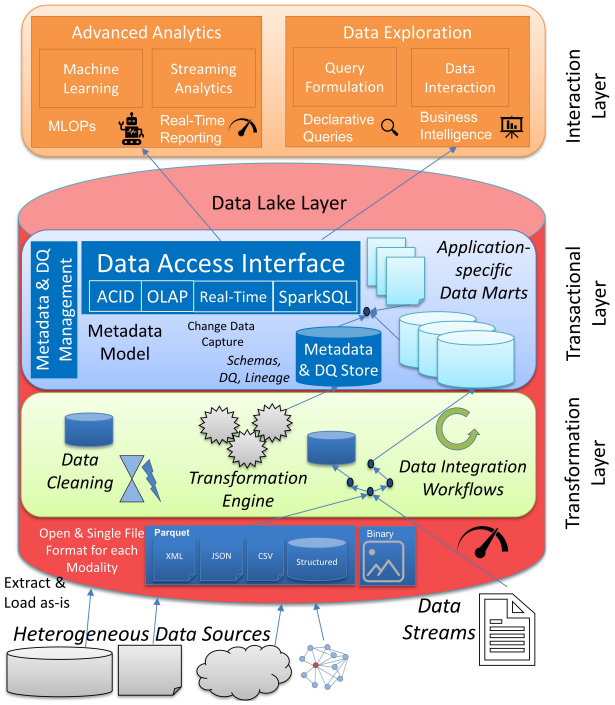


Figure 3.6: Data Lakehouse architecture inspired by [Jarke and Quix, 2017, Armbrust et al., 2021]

DLH4 — *ACID guarantees*: *provide means for checking and enforcing the consistency of structure and content across data collections, and ensure isolation and atomicity for all operations that modify or access the data concurrently.*

DLH5 — *Direct read access*: *provide unmediated, direct, and random read access to all (meta)data stored on the underlying storage system without needing to export and transform it first, and leverage open, standardized file formats.*

DLH6 — *Unified batch and stream processing*: *support processing in large-scale batches as well as at a high rate and in real-time."*

We derive a data lakehouse architecture (see figure 3.6) from the four-layered data lake architecture and the findings of Harby and Zulkernine [2022] and Azzabi et al. [2024]. In the figure, the ingestion layer is replaced by a composite *Data Lake Layer*. This layer is composed of an ingestion layer, to load data and extract metadata, as well as a storage layer. What differs is that the storage is composed of single scalable storage technology, for example *HDFS* or *Amazon S3*. This foundational layer serves as the primary storage for raw, structured, semi-structured, and unstructured data. It enables scalable, cost-effective data storage and accommodates diverse data sources, serving as a central repository for all ingested data. Further, the layer utilizes a single open data format (such as Parquet) for each modality.

The *Transformation Layer* is responsible for the ETL pipelines and thus dedicated to extracting data from various sources, transforming it into structured formats, and loading it for analysis. It standardizes and cleanses data to make it ready for analytics, enabling compatibility and consistency across data sources and preparing it for data access. *Transactional*

Layer supports data operations and guarantees ACID compliance, enabling reliable and transactional processing, often needed for complex analytical queries (OLAP) supporting accurate and real-time analytical processing. The transformation as well the transactional layer are embedded into the data lake layer to emphasize the vision that the lakehouse is a uniform system, and does not resemble a two-tier architecture composed of an individual data lake and data warehouse and thus no data replication or pipelines for transferring between the platforms are necessary. Finally, the *Interaction Layer* serves as the user interface ensuring that insights can be easily generated, accessed and utilized by business users and data scientists.

3.3.5 Data Fabric

The concept of data fabrics was coined by NetApp in a white paper from 2016 as a "vision for data management [...] that seamlessly connects different clouds, whether they are private, public, or hybrid environments" [Hechler et al., 2023]. Back then, the term meant a combination of different data storage and integration techniques, however, without details about a concrete architecture.

In 2019, Gartner defines a data fabric as a design concept for attaining reusable and augmented data integration services, data pipelines and semantics for flexible and integrated data delivery [DeBellis et al., 2023]. Hence, a data fabric is a technical architecture that brings together heterogeneous data that spans across multiple data sources; it allows organizations to monitor and manage data regardless of the location, considering appropriate data governance and data cataloging [Bode et al., 2024, Li et al., 2022, Abu Rumman and Al-Abbadi, 2023]. Then in 2021, Gartner [Thanaraj et al., 2021] redefines a data fabric as a "design concept that serves as an integrated layer (fabric) of data and connecting processes". In their data fabric architecture, Gartner emphasizes key characteristics, such as the use of AI for metadata management and data cataloging as well as knowledge graphs and ontologies to connect and enrich data sources with business related concepts. They introduce the notion of **"passive vs active metadata"**, i.e., data models, schema definitions, glossary, various logs are considered as passive metadata, whereas the continuous analysis of data usage with are described as active metadata describing actual experiences. The conversion of passive metadata into active metadata is interpreted as automating the collection and management of metadata [DeBellis et al., 2023]. By now, major vendors in data science have embraced the term; both vendors of semantic technology (e.g., Franz Inc., Ontotext, Pool Party, Stardog, and Top Quadrant) as well as more traditional vendors such as Microsoft, Oracle and Tibco [DeBellis et al., 2023].

Characteristics

At present, the particular mechanisms, standards, and technologies for the implementation of a data fabric architecture still have at be clarified as there is not a reference architecture or system [Hechler et al., 2023, Priebe et al., 2021, Abu Rumman and Al-Abbadi, 2023]. For example, some authors advocate for the presence of **data virtualization**, i.e., a software architecture that allows applications and end users to access data from multiple sources as if it were stored

in a single location, as a critical component of a data fabric [Kuftinova et al., 2022, AKERMI et al., 2023, Li et al., 2022]. In fact, the term "fabric" metaphorically represents the interwoven threads/fibers of a textile, signifying how different data sources, technologies, and workflows can be interconnected into a cohesive whole. As noted by *CData Software* [Software, 2024]: "*If we take the metaphor to its conclusion, data virtualization can weave the threads (data) to make the finished cloth (data fabric).*" Others see data virtualization as optional [Serra, 2024] or do not mention it at all [Macías et al., 2024]. In our requirements definition below, we consider data virtualization as a key requirement for data fabrics. Data fabrics should be used as an additional integration layer on top of other existing data platforms.

Similarly, various proposals include the use of knowledge graphs, ontologies and semantic web technologies explicitly [DeBellis et al., 2023] and others do not mention them at all [Rieyan et al., 2024]. DeBellis et al. [2023] provide strong arguments in favor of using semantic web technologies because they ensure reuse and interoperability of data.

Modeling metadata for a data fabric requires a complex, highly interconnected model that typically spans multiple organizational, geographic, and business boundaries. Obviously, relational data models are too rigid to easily store this type of data. Knowledge graphs are more suitable to represent such information because they are more flexible, heterogeneous, formal as well as fast and scalable [DeBellis et al., 2023]. Martinez-Casanueva et al. [2024] present *CANDIL*, a data architecture to convert heterogeneous data into an RDF-based representation using mappings. They call their platform a data fabric; however, besides the use of semantic data management techniques, we do not recognize other technical features in their approach. Therefore, we consider it as an ontology-based data integration pipeline.

A dominant theme in the data fabric paradigm is the idea of the **polystore**, i.e., combining heterogeneous storage databases, potentially hosted in the cloud [Priebe et al., 2021, Strengholt, 2023, Macías et al., 2024, Alvord et al., 2022, Moon et al., 2021]. Similarly, data storage is not meant to be centralized to a single location for access, but decentralized and distributed [Bode et al., 2024, Serra, 2024]. Data exploration and access is mediated by the fabric through centralizing metadata only [Rieyan et al., 2024, Li et al., 2022].

To facilitate this type of accessing, discovering, and understanding of data, but also for automating data integration and governance activities as well as privacy, and compliance-related topics, the data fabric puts metadata as the most critical element [Bode et al., 2024]. Hechler et al. [2023] differentiate the data fabrics paradigm from other data architectures by the predominant **use of AI to automate the metadata creation and management**, which is inline with the idea of active metadata.

According to Hechler et al. [2023], ensuring trustworthy AI by broadening the scope of traditional data governance toward unified AI governance is another requirement for data fabrics. This is a relatively new area that goes beyond the inclusion of AI artifacts into the governance realm, because it requires to detect bias, drift, and decreasing accuracy and precision and needs to propose corrective actions to realign AI artifacts to business goals.

Furthermore, so-called self-service capabilities play a critical role that allow users to independently perform tasks that would traditionally require IT experts. Hechler et al. [2023]

provide concrete examples: (1) access various assets through the knowledge catalog based on roles, without needing additional credentials, (2) metadata includes details about how to access assets (e.g., (No)SQL, REST APIs, etc.), (3) a holistic view of AI/ML assets (4) metadata generation through AI to automate tasks like resource allocation and semantic annotations, (5) semantic search uncovers correlations and recommends relevant assets for business purposes, (7) data curation tasks such as data quality assessment and refinement are manageable by users without expert intervention. These examples require the creation of standardized data pipelines based on comprehensive metadata management for the automated and systematic transformation, cleansing, and integration of data [Blohm et al., 2024, Moon et al., 2021, Macías et al., 2024].

An increasingly important pipeline is the support of the life cycle of AI processes [Hechler et al., 2023], i.e., the management of the in- & outputs and the creation of ML models. However, it is significant to note that, although researchers have recognized the potential [Abu Rumman and Al-Abbadi, 2023], more traditional BI and data science activities, such as reporting, modeling and analysis tools are not considered the core scope of a data fabric, but rather fall under the responsibility of the data consumers [Priebe et al., 2021].

For a definition we follow Blohm et al. [2024]:

Definition 5. [**Data Fabric**] A Data fabric is a data architecture "*for attaining reusable data integration services, data pipelines and semantics for flexible and integrated data delivery [Zaidi et al., 2019]. It builds on the analysis, creation, and usage of metadata [Priebe et al., 2021, Li et al., 2022], which are modeled as a knowledge graph.*" [Blohm et al., 2024]

Requirements

Li et al. [2022] identify key enabling technologies and Serra [2024] provide key features. Based on all the previous findings we provide technical requirements for data fabrics:

DF1 — Polystore and Decentralized Storage: ensures data can reside across data lakes, data warehouses, or other systems, reducing redundancy and enhancing scalability.

DF2 — Data Virtualization: abstracts data complexity allowing seamless access to distributed sources via real-time querying and integration without requiring a central data repository.

DF3 — AI-Driven Metadata Management: metadata management must be (semi-)automated using AI to generate, enrich, and maintain metadata dynamically. This supports efficient data discovery, lineage tracking, and governance, while reducing manual intervention.

DF4 — Self-Service Capabilities: standardized pipelines driven by metadata enable automated, systematic data transformation, cleansing, and integration, allowing users to perform exploration, curation, and analysis tasks in GUI-based workflows.

We introduce a data fabric architecture in figure 3.7, based on Hechler et al. [2023], Macías et al. [2024] and Priebe et al. [2021]. Heterogeneous data sources are first ingested into storage. We illustrate how the data fabric architecture utilizes various storage technologies in a decentralized polystore. Those may include entire data lakes, warehouses, file storages like HDFS or cloud storages like S3. As the fabric's main concern is not the storage itself, but

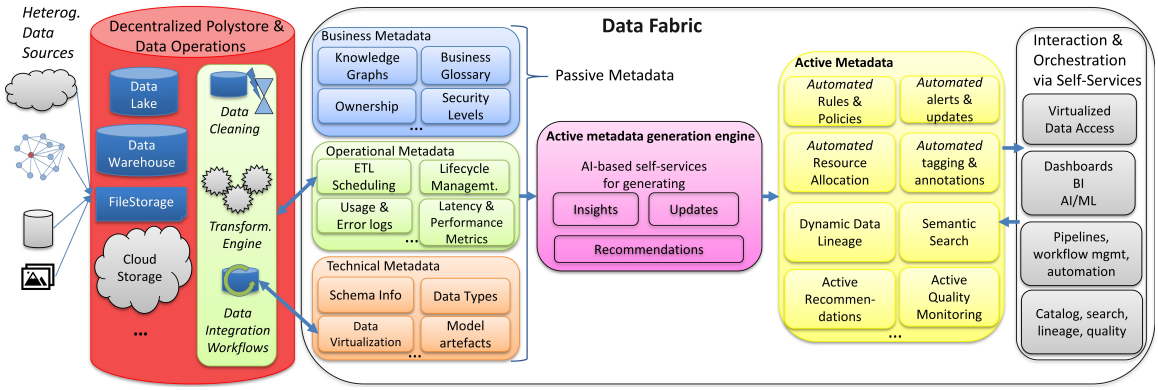


Figure 3.7: Data fabric architecture inspired by [Priebe et al., 2021, Hechler et al., 2023]

the associated metadata for merging the various parts into a unified view, we decided to leave this part out of the core data fabric area. Similarly, we view the transformation layer for data processing at scale as belonging to the storage component. The fabric is responsible for coordinating the required operations. However, it is ultimately the underlying storage, that is responsible to execute resource-intensive data processing operations. This opposes Priebe et al. [2021], who include storage and integration and interoperability into the fabric's core functionality. Even in their illustration one can make a case to divide a data fabric into a storage & operations component and a knowledge catalog section. Essentially, we view the data fabric as knowledge catalog that connects various underlying data storages through comprehensive metadata management capable of data access and transformation via predefined standardized data pipelines.

Following the illustration for active metadata by Hechler et al. [2023], for the data fabric we first deal with passive metadata divided into the three categories for metadata introduced by Diamantini et al. [2018]. Business metadata serializes the KGs and business or domain related knowledge. The operational metadata orchestrates ETL processes and other data processing operations on the underlying storage, collects logs and manages the data lifecycle. Technical metadata extracts schema information, data types, etc. Thereby, it enables data virtualization and collects metadata about the connections between analytical models and their artifacts and training datasets.

These are then converted into active metadata by the responsible generation engine. Those include tools powered by AI methods to enable users to discover insights, perform updates on the data, and receive recommendations. These AI methods are not meant to be crafted by users, but rather the data fabric should provide ready-to-use self-services that are sufficiently abstracted and simplified to make anyone capable of using them. The self-services enable a range of data discovery methods, such as semantic search, recommendations, automated semantic labeling, tagging, and annotations.

Finally, the interaction layer needs to provide an intuitive GUI that enables unfamiliar users to operate and utilize the self-services for integrating, searching and transforming data and orchestrating workflows.

3.3.6 Data Mesh

Centralized data teams, while highly specialized, often lack business domain expertise, creating bottlenecks in scaling data-driven decision-making across the enterprise [Butte and Butte, 2022, Blohm et al., 2024]. As different business units generate data in a decentralized manner, traditional architectures managed by central IT departments struggle to manage increasing volume and variety [Brackenbury et al., 2018, Dehghani, 2022, 2019]. On the other hand, if the organization is divided in multiple business units to manage datasets specific to their domain, a lack of transparency and communication can lead to replicated effort and data silos. Here, a self-serve platform and employing proper standards through federated governance may enable reuse of datasets by others. Furthermore, centralized data management raises concerns over data ownership and governance. Without clear responsibility for data quality and maintenance, organizations risk compromised data integrity. Shifting ownership back to the domains creates a strong sense of responsibility over the created data that will improve data quality [Dehghani, 2022, Bode et al., 2024].

For such reasons, Dehghani [Dehghani, 2019, 2022] introduces the so-called data mesh, which is not (yet) a typical data architecture with concrete implementations, or infrastructure specifications, but rather "a sociotechnical data approach" [Dehghani, 2022]. Interestingly, the concept draws inspiration from microservice architectures in software engineering [Ashraf et al., 2023]. While traditional data platforms remain centralized, a data mesh applies the microservice paradigm to data architectures, where each domain functions as an independent service. Just as microservices emerged to overcome the limitations of monolithic applications, the data mesh addresses the constraints of centralized data structures.

Another major motivating factor is the historical focus of data and analytics research on technological components rather than usage-driven approaches [Abbasi et al., 2016, Strengholt, 2023], which led to overlooking the significance of federated architectures, governance, and organizational design [Blohm et al., 2024]. This is the main reason for bottlenecks in centralized data platforms. Most studies have centered on designing technical solutions rather than addressing the organizational, procedural and governance challenges inherent in large-scale data management. Machado et al. [2021] even go as far as calling the data mesh a paradigm shift in the field of data architectures. In this new paradigm, data is seen as the main concern of a truly data-oriented organization, and the pipelining tools and specific architectures itself are seen as a secondary concern.

Several publications mention a close relationship between the data fabric and the data mesh [Hechler et al., 2023, Strengholt, 2023, Blohm et al., 2024, Serra, 2024, Bode et al., 2024]. Serra [2024] describes data fabrics as an architectural and technological framework that operates within or across domains in a data mesh, supporting its components and enabling synergy rather than competition. While a data mesh offers a decentralized, domain-oriented blueprint, data fabrics provide the underlying infrastructure to implement it. Multiple data fabrics can coexist within a mesh, and this integration approach also extends to other data architectures like data warehouses or lakehouses.

Despite the theoretical foundation of a data mesh and related frameworks, research on their real-world application remains limited. While studies have explored technological architectures, privacy concerns, and individual case studies, there is little systematic analysis of how industry implements these concepts [Bode et al., 2024]. However, Machado et al. [2021] outline two industrial use cases:

Zalando [Schultze and Wider, 2020], a leading European fashion platform, initially relied on a centralized data lake but faced challenges such as lack of data ownership, poor data quality, and scalability bottlenecks as data sources and consumers grew. To address these issues, the company implemented a data mesh, introducing decentralized data ownership, domain-oriented teams, and a shift toward treating data as a product rather than a by-product. This transition enabled interoperability and decentralizing data responsibilities and improved governance through a metadata layer.

Netflix [Cunningham, 2020, Pires et al., 2023, Nguonly et al., 2021], serving over 150 million global users, generates petabytes of data daily and sought to integrate its studios into a unified system capable of handling this massive volume efficiently. Due to redundant pipeline efforts, maintenance overhead, lack of best practices, high latency, and error correction issues, Netflix implemented a data mesh that abstracts pipeline complexity for users. The infrastructure allows users to build pipelines while accessing a metadata catalog and standardized processes without needing deep technical knowledge.

Characteristics

Four principles define the data mesh as originally introduced by Dehghani [Dehghani, 2019]:

DM1 — Data-as-a-product: creation, management, and distribution of data products via a catalog. Data providers are responsible for publishing and provisioning them and collaborate with data owners and stewards for maintenance.

DM2 — Decentralized data ownership and architecture: each domain has ownership over its data products, ensuring that domain teams are responsible for the quality, maintenance, and lifecycle of their data products.

DM3 — Self-Service capabilities: a self-service data platform must be established to enable domain teams to create, manage, and consume data products without requiring extensive technical expertise.

DM4 — Federated governance: a federated governance model balances decentralized autonomy with global governance standards, ensuring that policies are enforced consistently across all domains while allowing flexibility for domain-specific rules.

The data-as-a-product principle frames data products as deliberately distributed and interconnected across individual “mesh nodes” [Machado et al., 2021]. To give an example for a data product, consider the *MLFlow* model export functionality that outputs not only the trained model parameters, but also essential metadata, dependencies, and governance information. Each exported model includes environment specifications such as a `requirements.txt` or `conda.yaml`, ensuring reproducibility across different infrastructure setups. Additionally, *MLFlow* tracks ex-

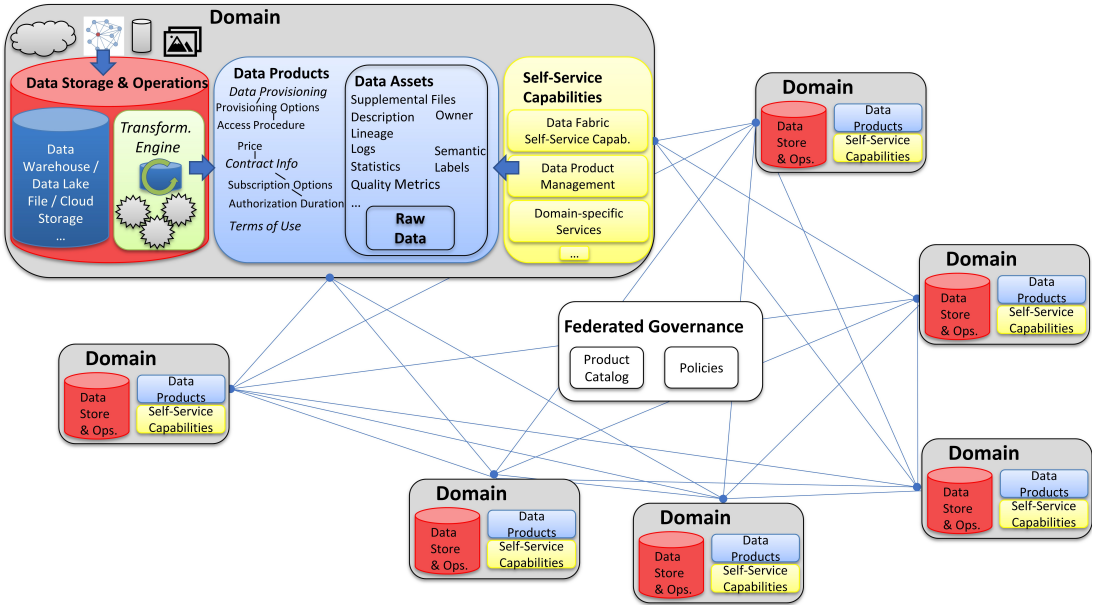


Figure 3.8: Data mesh architecture inspired by [Eichler et al., 2022, Dolhopolov et al., 2024, Hechler et al., 2023, Strengholt, 2023]

periment metadata, including training hyperparameters, dataset versions, and performance metrics, allowing for model lineage and governance.

Eichler et al. [2022] recognize that data providers often lack incentives to share their data, because they face additional effort without immediate benefits, discouraging participation. To address this, they subdivide data products further into data assets. Data assets are raw, registered data in a company's catalog, while data products are refined versions with defined metadata, usage policies, and provisioning options. The transformation process follows three scenarios: (1) a provider directly registers data-as-a-product, (2) a consumer requests access to a data asset, prompting its transformation, or (3) a steward or another employee enriches metadata and submits it for approval. This staged approach reduces the initial burden on providers, ensuring they only need to add metadata when data is actually requested.

Domain-oriented ownership shifts data responsibility to individual business domains, allowing them to manage and utilize their own data according to their specific operational logic. This model fosters agility and autonomy by structuring data management around product-focused teams [Machado et al., 2021, Bode et al., 2024]).

Self-service platforms serve as a technical abstraction layer, enabling unified access to data products while reducing complexity for users [Wider et al., 2023, Bode et al., 2024]. By providing a standardized infrastructure, they eliminate redundant technical efforts and simplify the management of data products across their lifecycle [Dehghani, 2022, Bode et al., 2024, Araújo Machado et al., 2022].

Finally, federated computational governance establishes global governance standards that ensure interoperability and secure data access while enabling decentralized and autonomous governance by distributed teams. By balancing centralized standards with decentralized decision-making, federated governance ensures that data remains accessible and governed in a struc-

tured yet flexible manner [Dehghani, 2022, Bode et al., 2024, Wider et al., 2023]. Policies are also designed to balance local, i.e., domain-oriented, autonomy with global standardization. The federated governance model oversees security policies (defining authentication, authorization, and access control), interoperability policies (standardizing schema definitions, data contracts, and API structures), communication policies (guidelines for cross-domain data sharing), and documentation policies (requiring metadata enrichment, lineage tracking, and accessibility compliance)

For a definition we follow Blohm et al. [2024] once more:

Definition 6. [**Data Mesh**] "*Data mesh is a socio-technical, decentralized, distributed concept for enterprise data management [Bode et al., 2024]. It is characterized by the four principles of data-as-a-product, domain orientation, self-service platforms, and federated computational governance [Dehghani, 2020].*"

Figure 3.8 illustrates the data mesh architecture. van der Werf et al. [2024] identify four archetypes, out of which the fine-grained fully federated mesh or short pure data mesh is considered to be the most theoretical and decentralized but also the most mature version. We have chosen this pure form in which individual nodes directly exchange data products and deliberately left out a central distribution layer. Every atomic team or business unit is responsible for a single domain and the data products arising from it. While this approach provides high flexibility and strong domain specialization, it might also introduce challenges such as wasting resources, when data transformations are performed redundantly due to lack of communication or transparency between domains. Here, the maturity of the self-service capability and federated governance components are particularly relevant. In the figure, one node is enlarged representatively to display more details. In each domain heterogeneous raw data emerges that will be stored in a data storage and operations platform of choice. Those can be whatever is available and suitable in the specific domain, e.g., data warehouse/lake, cloud storages or file storages etc. Each domain is responsible for transforming and packaging raw data into data products. Data products are further subdivided into data assets and raw data as inspired by Eichler et al. [2022]. In each domain, data consumers and providers have access to self-service capabilities. In the figure, the individual nodes inherit self-service capabilities from a data fabric to represent the close relationship and potential synergy between the two paradigms. They are extended by domain-specific services as well as data product management where domain teams register their data products along with associated metadata, quality metrics, and access policies and facilitate the distribution to other domains. This data product management and distribution is overseen by a central data federated governance layer to enforce global governance standards consistently ensuring policies, interoperability and secure data access.

3.4 Evaluation

The evaluation begins with a definition of the criteria used to assess and describe the discussed architectures. The criteria are defined in a way that allows to assign a quantitative measure (see table 3.1). Finally, these evaluation criteria in conjunction with the technical requirements from

section 3.2 are evaluated against real-world systems to find a suitable match.

3.4.1 Dimensional Analysis

The preceding literature review and the in-depth examination of the discussed data architectures helps us to derive the following dimensions. These aim to (a) describe the characteristics of a given architecture, (b) highlight similarities and differences between them, and (c) provide a framework for assigning existing systems to one of the seven categories in section 3.2:

Control, Governance & Trust

D1 – Architectural Paradigm: Measures how centralized or decentralized the data architecture is in terms of control, storage infrastructure, and processing. Higher scores reflect distributed, domain-driven models that promote autonomy.

D2 – Governance & Ownership: Examines the model of data control from centralized to federated ownership. Higher values indicate domain-level accountability and governance.

D3 - Security & Trust Measures how much the architecture gives priority to support secure access, data protection, auditability, and trust-building mechanisms such as data provenance, policy enforcement, or verifiable sharing.

Data Modeling & Understanding

D4 – Data Formats: Evaluates how much the architecture considers flexibility in handling different data formats. Higher scores indicate native support for polystores and unstructured, semi-structured, and structured data.

D5 – Metadata Management: Assesses how metadata is handled from manual and static to dynamic, automated, and lineage-aware systems. Advanced setups foresee the use of AI or active metadata.

D6 – Knowledge Management: Assesses the extent to which the architecture supports modeling, managing, or leveraging domain knowledge alongside data. This includes formal approaches (e.g. SDM) and informal ones, such as business glossaries, tagging systems, or f.e. enterprise LLM-based chatbots [Yun et al., 2025] with richer data understanding, reasoning, and integration.

Data Processing

D7 – Integration Focus: Captures how central integration is to the architecture's design. A high score means data integration is a foundational concern rather than an afterthought.

D8 – Data Virtualization: Describes how much focus the architecture places on abstracting storage details to enable querying flexibly and transparently without having to move data physically.

D9 – ML & Analytics Support: Captures the degree of support for ML and data science. Higher scores indicate built-in tools and pipelines for advanced analytics beyond traditional BI.

#	Dimension	1 = Low	3 = Medium	5 = High
Control, Governance & Trust				
D1	Architectural Paradigm	Centralized	Federated	Decentralized
D2	Governance & Ownership	Centralized	Shared or hybrid	Federated / Domain-oriented
D3	Security & Trust	Minimal/static security	Moderate control with some auditing	Integrated, policy-enforced, trust-aware
Data Modeling & Understanding				
D4	Data Formats	Structured only	Multi-format, some flexibility	Polystore / All formats
D5	Metadata Management	Manual / fixed	Automated extraction / tagging	AI-driven, dynamic, lineage-aware
D6	Knowledge Management	None	Basic domain modeling (glossaries, tags, rules)	Integrated knowledge layer (ontologies/KGs, LLMs)
Data Processing				
D7	Integration Focus	Not central to the architecture	Localized or externalized	Core architectural concern
D8	Query Mechanisms	Local SQL only	SQL + NoSQL / partial federation	Federated / virtualized querying
D9	ML & Analytics Support	BI only	Some support for ML	Full ML pipeline and data science support

Table 3.1: Evaluation Dimensions for Data Architectures on a scale from 1 to 5, mapped to General Requirements

3.4.2 Evaluation of Real-World Systems

The Stackable Data Platform Stackable[1] is an open-source modular data platform built on Kubernetes that integrates a curated set of big data and data science tools into a cohesive platform-as-code architecture. It provides production-ready deployments of components like Kafka, Trino, Open Policy Agent, Hive via Kubernetes operators, with strong support for declarative configuration and infrastructure management.

From a control and governance perspective (D1–D3), Stackable promotes a hybrid governance model. While orchestration and policy enforcement can be centralized (e.g., via Kubernetes and Open Policy Agent), the platform enables domain-level autonomy in configuring and managing services, fostering decentralized ownership. It supports advanced access control through integration with offering a relatively mature security and trust enforcement model. In terms of data modeling and understanding (D4–D6), Stackable is tool-agnostic and does not prescribe a specific data model. It supports diverse data types through connectors and engines, but leaves semantic enrichment and metadata management to external services. Similarly, ontologies and knowledge graphs are not natively integrated but can be layered in through additional tooling. Regarding data processing (D7–D9), Stackable enables large-scale data integration and analytics through its orchestration of distributed compute engines like Spark and Flink, and supports virtualization and query federation via Trino. However, it lacks global semantic abstraction and unified orchestration of ML workflows. While the platform supports full ML pipelines via integrations, MLOps is not a primary design focus.

Stackable does not conform strictly to data lakehouse or data fabric paradigms due to its lack of unified metadata, active governance layers, or enforced semantic models. However, it aligns most closely with **data mesh** principles particularly through its domain-oriented deploy-

[1] https://stackable.tech/, accessed on 02.05.2026

	Data Warehouse [Quix, 2003, Inmon, 2005, Bose, 2009]	**Data Lake [Hai et al., 2023, Quix and Hai, 2019, Azzabi et al., 2024, Nargesian et al., 2019]**	**Semantic Data Lake [Hoseini et al., 2024d, Dibowski et al., 2020, Bagozi et al., 2019]**	**Data Lakehouse [Harby and Zulkernine, 2022, Schneider et al., 2024, Armbrust et al., 2021]**	**Data Fabric [Hechler et al., 2023, Blohm et al., 2024, Serra, 2024]**	**Data Mesh [Dehghani, 2022, Serra, 2024, Machado et al., 2021]**
Control, Governance & Trust						
D1	1 – Centralized relational	1 – Centralized raw storage	1 – Semantic layer on centralized DL	1 – Hybrid of DW & DL	3 – Federated integration & Active Metadata	5 – Decentralized do-main–oriented
D2	1 – Centralized	1 – Centralized	1 – Centralized	1 – Centralized	3 – Pol-icy–managed	5 – Domain ownership
D3	2 – Strong role–based security but typically centralized and static	1 – Security is often minimal or bolted–on; access control at storage layer	2 – semantic technologies to account for it, but not a primary focus	3 – security models via modern engines, but not the primary focus	5 – central	4 – Domain–level responsibility; requires strong governance frameworks to scale
Data Modeling & Understanding						
D4	1 – relational	5 – raw data in original format	5 – raw data in original format	3 – Mainly relational	5 – arbitrary format	5 – Data products of arbitrary format
D5	1 – Fixed, scoped model	3 – Automated metadata extraction	4 – Metadata enriched with semantics	3 – Automated metadata extraction	5 – AI–driven, automated	2 – Decentral-ized, on–demand
D6	1 – None	1 – None	5 – Ontologies & KGs	3 – Basic	5 – KG–based metadata	2 – local vocabularies, but no global semantic layer
Data Processing						
D7	5 – ETL	3 – on–demand integration (ELT)	4 – Semantic–level integration	2 – Mix between DL and DW	4 – Integration driven by metadata	2 – Local integration within data products
D8	2 – access remains tightly coupled to internal engine and physical schemas	3 – Raw file-based querying; flexibility via tools like Spark/Presto, but limited abstraction	5 – OBDA-based access over heterogeneous data using semantic models.	3 – Combines files and tables with SQL engines, but limited abstraction from physical storage	5 – virtualized access, logical layers, and cross-platform queries	3 – Varies by domain; considers APIs or connectors; lacks global query abstraction
D9	1 – Focus on reporting & BI	2 – Not the primary focus	1 – No focus	5 – Full ML lifecycle support	3 – Moderate	1 — Not a focus

Table 3.2: Evaluation of Data Architectures against requirements

ment patterns (DM2), emphasis on infrastructure-as-code, and federated governance (DM4). While it lacks the organizational enforcement, its architectural modularity and governance extensibility make it a strong enabler for infrastructure-centric data mesh implementations. If these are extended by self-service capabilities (DM3) in the future, possibly by integrating corresponding external services, then Stackable covers a mature technological base for data meshes.

Microsoft Fabric Microsoft Fabric[2] is Microsoft's platform-as-a-service offering, unifying components from Power BI, Azure Synapse, Azure Data Factory, and OneLake. Designed for end-to-end data and AI workloads, it provides services for data engineering, data science, real-time analytics, and business intelligence — with strong emphasis on cloud-native scalability, collaboration, and ease of use.

From a control and governance perspective (D1–D3), Microsoft Fabric implements a federated orchestration model layered over centralized infrastructure. While the platform itself is centrally managed (e.g., via Azure), governance can be delegated across workspaces and domains. It offers comprehensive role based access management (RBAC), lineage tracking, and sensitivity labeling features. Data access and governance is handled by Microsoft Purview, though policy-based data sharing is still evolving. In terms of data modeling and understanding (D4–D6), Fabric supports a wide range of data formats promoting a polystore architecture via OneLake, where all data can be virtually accessed through a unified namespace. Metadata extraction and enrichment are automated and AI-augmented, particularly through Microsoft Copilot. While domain modeling support exists (via data cataloging, tagging, glossaries), deeper semantic constructs such as ontologies or knowledge graphs are not first-class features. Regarding data processing (D7–D9), Fabric offers highly integrated support for ETL pipelines, real-time streaming, and analytical processing. It supports distributed querying across files and tables, but semantic virtualization and ontology-based abstraction are not core design elements. Fabric fully integrates AutoML and MLOps capabilities enabling full ML lifecycle support from within the platform.

Microsoft Fabric aligns closely with the **data fabric** paradigm (DF1, DF3 & DF4), particularly in its unified infrastructure, active metadata, and orchestration of cross-role workflows. While data virtualization can be configured via so-called *shortcuts*, it falls short of the semantic richness for advanced data virtualization (DF2). Overall, Microsoft Fabric represents a commercially mature, enterprise-oriented fabric implementation that balances manageability and innovation, with strong vertical integration across Microsoft's data ecosystem.

3.5 Related Surveys & Work

The ISO/IEC TR 20547-1:2020 standard [International Organization for Standardization, 2020] defines a reference architecture for big data systems, providing a high-level framework to guide the design of interoperable and scalable solutions. While DATER shares the goal of architec-

[2] https://www.microsoft.com/en-us/microsoft-fabric, accessed on 02.05.2026

	SEDAR [Hoseini et al., 2023]	Stackable [Team, 2025]	Microsoft Fabric [Ghosh, 2024]
Control, Governance & Trust			
D1	1 – Centralized	5 – Decentralized	3 – centralized governance, but workspace-level autonomy & logical domains
D2	1 – Centralized control plane	4 – centrally policy-managed, but configurable from within domains	3 – centralized governance tools, but support for role- and workspace-based ownership models
D3	1 – Basic RBAC; not a strong focus	4 – sophisticated and scalable measures	3 – fine-grained access-policies, but policy-based data sharing are not core.
Data Modeling & Understanding			
D4	3 – Raw data retained; main support relational	3 – arbitrary data can be processed, but support mainly designed for relational model	5 – services integrate a variety of formats suitable for polystore-style needs.
D5	5 – Automatically enriched with semantics	1 – Outsourced to external tooling	5 – AI-augmented, consistent with modern active metadata
D6	5 – Ontologies & Knowledge Graphs	1 – None	3 – basic domain modeling capabilities such as tagging, schemas, and glossaries.
Data Processing			
D7	5 – Integration is central	2 – ingestion and ELT workflows, integration is often manual or tool-driven	5 – Integration is central
D8	5 – OBDA	3 – virtualization via Trino connectors, but no global query abstraction	4 – can be configured via OneLake, but no global semantic layer
D9	5 – Full ML lifecycle (incl. AutoML)	2 – possible, but not a special focus	5 – Full ML lifecycle

Table 3.3: Evaluation of real-world data architecture systems based on technical requirements.

tural classification, it places greater emphasis on comparative evaluation across diverse architectures based on concrete technical requirements. A whitepaper by Fraunhofer ISST explores the conceptual and technical intersections between data mesh and data spaces [Jahnke et al., 2023], highlighting their complementary nature in decentralized data ecosystems. It argues that data mesh offers organizational and governance principles within enterprises, while data spaces provide the infrastructure for secure, cross-organizational data sharing. Priebe et al. [2021] present architecture diagrams in ArchiMate notation [Josey, 2019] for data warehouse, data mesh, and data fabric based on the DMBOK [International, 2017]. Atzori et al. [2024] compare data spaces to data warehouses, lakes and different types of database systems.

In his book [Serra, 2024], James Serra gives a high-level introduction into modern data architectures: data warehouse, lake, lakehouse, fabric and mesh. As a textbook, it is more oriented towards readers unfamiliar with the field, explaining many concepts in much detail,

which we assume to be available with readers of this article. We draw inspiration from this work heavily but shift the focus towards more in-depth technical detail, comparisons, and add more architectures.

A systematic literature review by Ataei and Litchfield [Ataei and Litchfield, 2022] reveals that many data architectures rely on monolithic data pipelines with central storage, which hinders scalability and maintainability. Moreover, they point out that most existing reference architectures align closely with traditional data warehousing concepts, often integrating data lakes in a way that does not fully address modern data processing challenges. A major conclusion from their work is the necessity for decentralized and distributed big data architectures suggesting that emerging paradigms such as data mesh could better support scalability, data ownership, and interoperability as compared to centralized architectures. However, they note that industry adoption of such architectures remains slow due to the high cost and complexity of implementation. Additionally, the study identifies a lack of rigorous evaluation methodologies, making it difficult to assess their effectiveness in practice. Our work builds upon these insights by providing a more in-depth technical comparison of modern data architectures.

3.6 Conclusion

In response to the growing complexity of modern data landscapes and to address RQ1, this chapter provides a structured exploration and comparison of contemporary data architectures.

The study reveals that no single architecture universally satisfies all technical needs, underscoring the importance of context-aware selection and hybrid solutions. This led to convergence of architectural paradigms, e.g., lakehouses and fabrics blending the strengths of lakes and warehouses. This comes with a lack of consensus in the academic and industrial literature regarding the definitions and boundaries of and between certain architectures, indicating a need for a clear characterization of data architectures that motivated this work.

Modern data ecosystems increasingly shift toward decentralization, which promote domain-oriented ownership, federated governance, and peer-to-peer data sharing. The increasing importance of metadata and semantic technologies across architectures, highlights another shift toward more intelligent, discoverable, and machine-interpretable data systems. Architectures like data mesh and data fabric reflect more than technical designs. They also embody operational principles, organizational models, and mark a third cultural shift in data management.

3.6. CONCLUSION

4

The Semantic Data Reservoir (SEDAR)

> "*Those who have the good fortune to be able to devote themselves to scientific pursuits must be the first to place their knowledge at the service of humanity.*"
>
> Karl Marx [Lafargue, 1890]

In this chapter, we propose a data lake system architecture and give a comprehensive introduction to its workflow and components, as the solution for RQ2. In the previous chapter, we have seen the conceptual blueprint for data lakes given by the DATER framework. Here, we hope to provide a practical view of a concrete implementation, called the Semantic Data Reservoir (SEDAR) suitable as a workbench for further research on data architectures.

The SEDAR architecture constitutes the foundations upon which a substantial part of the subsequent research contributions are grounded. Those contributions extend SEDAR beyond a classical data lake and are presented in detail in chapters 5 to 7.

The development of SEDAR originated from the practical challenges encountered in the i^2DACH project (section 1.1), where diverse datasets from industrial chemistry had to be ingested, interpreted, linked, and analyzed over extended periods of time. While the system emerged from concrete application needs, its primary contribution lies in abstracting these experiences into a reusable architectural design that generalizes beyond a single domain or deployment. SEDAR addresses, in combination, the challenges of heterogeneity, semantic interpretation, analytics integration, and usability in industrial data ecosystems; these challenges are typically treated in isolation or only partially met by existing systems.

An early version of SEDAR was first published at CIKM'23 [Hoseini et al., 2023], where its core principles were presented.

Chapter Outline

This chapter is structured as follows:

- Section 4.1 introduces the research motivation and architectural context.
- Section 4.2 derives design objectives and research requirements.
- Section 4.3 describes the concrete implementation along the four core layers:
  - ingestion (section 4.3.1),
  - storage (section 4.3.2),
  - transformation (section 4.3.3), and
  - interaction (section 4.3.4),

4.1 Motivation and Research Context

This chapter describes the *core architectural framework* of the dissertation upon which a substantial part of the subsequent research contributions and publications are grounded. Later chapters extend the core architecture with semantic data management, automated analytics, and natural-language interaction (chapters 5 to 7). Accordingly, this chapter focuses on establishing the architectural baseline that enables those extensions.

Early on, SEDAR followed the paradigm of a *vanilla data lake*, prioritizing scalable storage, flexible schema-on-read access as well as data ingestion, governance and metadata management at scale. While this approach enabled rapid ingestion and experimentation, it quickly revealed fundamental limitations: semantic interpretation was external to the system, and analytical workflows lacked reproducibility and governance.

Existing data-management architectures address subsets of these challenges in isolation. Classical data lakes focus on scalable storage but provide limited support for semantic interpretation, and user-centric access [Hai et al., 2023, Tang et al., 2024]. Knowledge-graph–centric approaches excel at semantic integration but often remain detached from large-scale data ingestion and heterogeneous storage. More recent paradigms such as data fabrics propose integrated solutions, yet frequently remain conceptual or tied to vendor-specific ecosystems.

The motivation behind SEDAR is to overcome these limitations by evolving beyond a vanilla data lake toward a *semantic data reservoir*: an architecture in which data ingestion, storage, metadata management, semantic interpretation, and analytics are treated as interdependent components of a unified lifecycle.

From a research perspective, SEDAR serves as an experimental and integrative platform. Rather than optimizing for a single application scenario, the architecture was deliberately designed to support architectural evolution and comparative analysis. This makes SEDAR suitable not only as a production system but also as a research artifact that enables the systematic study of semantic technologies, automated analytics, and interaction paradigms under realistic industrial constraints.

4.2 Research Objectives

The design of SEDAR is guided by research objectives that operationalize the dissertation's research questions as *architectural hypotheses*: each objective states a property that an industrial data architecture should satisfy in order to remain usable under heterogeneity, evolution, and semantic ambiguity. These objectives are informed by both theoretical insights and empirical findings on ingestion and metadata management in heterogeneous industrial environments [Sawadogo and Darmont, 2021, Eichler et al., 2020, Liu et al., 2020a]. While SEDAR realizes a four-layer data architecture (ingestion, storage, transformation, and interaction), the primary focus of this chapter is on the ingestion layer and the data catalogue as the foundations of semantic integration and user-facing access.

Primary objective: Unified, ontology-mediated data access. The central objective is to enable *unified data access* across heterogeneous, evolving data sources. In the spirit of ontology-based data access (OBDA), SEDAR treats a shared conceptualization (ontology/vocabulary) as an *access contract*: users and downstream components interact with data through a stable, conceptual interface, while mappings and metadata mediate between this interface and the concrete representations of sources. The research aim is not a specific query engine or storage technology, but an access abstraction whose behavior can be stated and assessed independently of physical storage, source formats, and schema idiosyncrasies. This objective directly supports long-term usability and forms a prerequisite for later chapters on automated analytics and natural-language interaction, which rely on stable semantics and reproducible access to evolving datasets.

Enabling objective: Source- and format-agnostic data acquisition. Prior work on data lakes highlights that tight coupling between ingestion logic and concrete formats or systems severely limits extensibility and reuse [Jarke and Quix, 2017, Quix et al., 2016]. Accordingly, SEDAR frames ingestion as a *technology-independent acquisition abstraction* in which data acquisition is specified declaratively and can be instantiated for different sources without changing the conceptual contract of access. This objective elevates ingestion from an implementation detail to a research-relevant architectural concern: it enables ingestion strategies to be compared and evaluated under a common abstraction, thereby supporting architectural generality and comparability as required by RQ1.

Enabling objective: Continuous ingestion, lifecycle management, and change-aware versioning. Literature on metadata-management stresses that insufficient support for continuous ingestion undermines data freshness and downstream usability, particularly in environments with frequently changing data sources [Sawadogo and Darmont, 2021, Eichler et al., 2020]. SEDAR therefore models ingestion as a lifecycle-aware process and couples it to explicit representations of dataset versions and deltas. Such mechanisms are widely recognized as prerequisites for reproducible data analytics [Labio and Garcia-Molina, 1996, Armbrust et al.,

2020]. From a research perspective, change-aware versioning is an enabling condition for unified access: it makes the semantics and results of access *time- and version-relative* rather than implicitly dependent on a moving operational state. This objective addresses the integration challenges formulated in RQ2 and establishes a foundation for subsequent semantic integration and usability improvements in RQ3.

Enabling objective: Metadata-first and multi-granular semantic cataloguing. Multiple studies identify metadata quality as a central success factor for data lakes and data catalogues, directly influencing data discoverability, integration, and reuse [Nargesian et al., 2019, Hai et al., 2023, Scholly et al., 2021, Quix et al., 2016]. Accordingly, SEDAR adopts a metadata-first principle: no dataset is considered available without a corresponding metadata representation that captures structure, descriptive context, and provenance. Beyond dataset-level descriptions, existing solutions are often limited in their ability to represent deeper structural and semantic relationships [Liu et al., 2020a, Eichler et al., 2020]. To enable ontology-mediated access under domain-specific constraints, the catalogue therefore operates at multiple levels of granularity (datasets, entities, attributes, and nested structures), and it explicitly supports semantic linking and structural variation (data polymorphism). This objective provides the basis for interoperability and knowledge-driven integration as formulated in RQ3, while remaining grounded in the operational requirements captured by RQ2.

4.3 Implementation

SEDAR's architecture follows a microservice-based, modular, and extensible design. Built on existing open-source technologies in the area of big data management, it is suitable as a workbench for further research in data lakes. The **core of the system** is illustrated in figure 4.2 and based on an architecture with four layers [Jarke and Quix, 2017]:

1. The **ingestion layer** (section 4.3.1) is responsible for acquiring, harmonizing, and versioning data from diverse systems.
2. The **storage layer** (section 4.3.2) implements a polystore abstraction [Khan et al., 2019] to support heterogeneous structures without enforcing a single physical model.
3. The **transformation layer** (section 4.3.3) implements various operations for data preparation, e.g., cleansing, transformations, or joins.
4. The **interaction layer** (section 4.3.4) provides user-facing interfaces to support both expert and non-technical users.

A fundamental observation is that heterogeneous industrial data requires a unified computational environment capable of handling structured, semi-structured, and unstructured data at scale. This requirement is met by a **distributed processing engine** called *Apache Spark*. Spark's ability to process relational, nested, and streaming data through a common DataFrame abstraction highlighted the value of a uniform logical representation across heterogeneous data structures, reinforcing insights from RQ2 concerning industrial chemistry's data diversity.

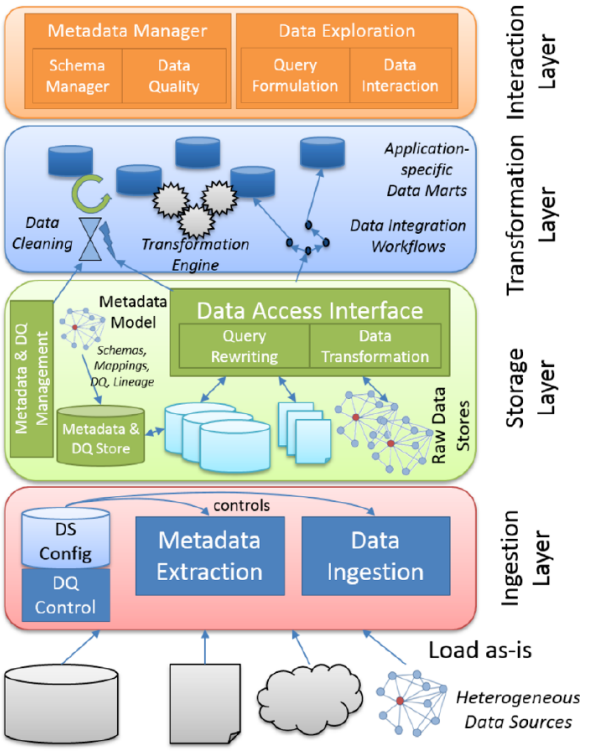


Figure 4.1: Conceptual four-layered data lake architecture from section 3.3.2

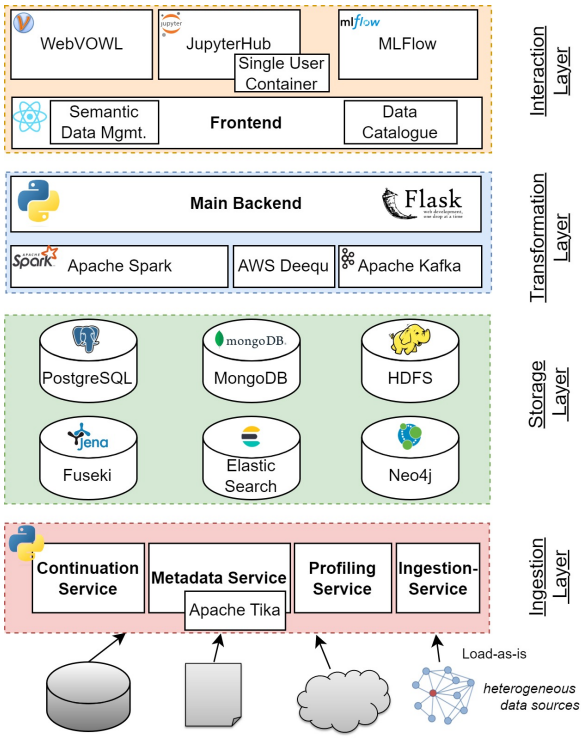


Figure 4.2: SEDAR's architecture [Hoseini et al., 2023], as a concrete implementation of the four-layered data lake architecture.

Similarly, the system employs multiple storage backends, including a **relational database** (*PostgreSQL*), a **document store** (*MongoDB*), and a **distributed file system** (*HDFS*).

On the interaction side, the system implements a **client–server architecture** with a web-based interface backed by a lightweight service framework (*Flask*). Conceptually, interaction logic must remain decoupled from data-management logic to support a broad range of user roles, including non-technical domain experts (RQ4).

Deployment via Microservices A core architectural decision in the design is the decomposition of SEDAR into independent microservices. In the context of RQ2, this provides main architectural advantages: (1) loose coupling of system responsibilities, (2) independent scalability and (3) portable and reproducible deployment. To operationalize these architectural principles, the implementation of SEDAR adopts a container-based deployment model providing lightweight, isolated execution environments by bundling each microservice with its runtime dependencies.

4.3.1 Ingestion Layer

In the early stages of the system, data ingestion depended heavily on the data source and the selected target storage and thus had issues with generality. First, users needed to manually select a target storage system from the available options, for which compatibility problems can

arise if the data source format is incompatible with the chosen storage. Then, the processing logic for transforming data from sources to storage was hardcoded into the server. This made it impossible to adapt to new data formats. Therefore, a dedicated ingestion interface should be developed for the system, which was initially designed, developed and implemented by Martin [2022] and continually progressed to incorporate evolving requirements. In particular, Thiel [2022] added a framework for comprehensive metadata extraction to populate the system's data catalogue with corresponding metadata. The design of the ingestion layer revolves around five key principle.

Key principles

(1) Format-independent and extensible ingestion: Industrial environments routinely produce data in diverse and evolving formats, ranging from CSV and relational exports to deeply nested JSON, XML, sensor streams, and proprietary binary structures. To avoid constraining the system to a fixed set of supported formats, SEDAR adopts a format-agnostic ingestion abstraction, in which data sources can be registered without prior assumptions about their structure. This abstraction is technology-agnostic but is practically enabled through a distributed processing engine (represented in our implementation by Apache Spark). Spark's DataFrame model served to demonstrate that a uniform logical representation across heterogeneous formats is viable and scalable.

(2) Storage independence and seamless integration with a polystore: Because SEDAR employs a polystore storage layer (see section 4.3.2), ingestion cannot be tied to a single physical storage model. Instead, the ingestion architecture abstracts from storage details and allows datasets to be directed to different storage engines according to structural properties, lifecycle needs, or user-defined policies. This design principle ensures that SEDAR can naturally accommodate:

- structured datasets from relational databases,
- nested or document-oriented data,
- unstructured files,
- versioned or raw-storage datasets.

(3) Continuous ingestion and support for streaming data: Many industrial datasets do not arrive as one-time uploads but as continuous streams or scheduled updates (e.g., laboratory instrumentation, sensor measurements, recurring ERP exports). To address this, the ingestion layer adopts a lifecycle-oriented orchestration model, enabling:

- periodic ingestion tasks based on declarative schedules,
- continuous streaming ingestion,
- automatic resumption of ingestion tasks,
- and decoupling between ingestion logic and user-triggered events.

A message-oriented communication backbone (represented in our implementation by *Kafka*)

provides the architectural capability for asynchronous and scalable coordination across ingestion-related services. Conceptually, this ensures more robust execution than monolithic or tightly coupled designs.

(4) Dataset versioning and change-data capture Updates to industrial datasets frequently introduce new records, modify existing ones, or remove obsolete entries. To support reproducibility, data provenance, and longitudinal analysis, the ingestion layer treats dataset versioning as a first-class concern. Each ingestion event produces a new dataset version, with mechanisms for computing incremental changes (deltas). This principle provides several conceptual benefits:

- historical states can be reconstructed at any time,
- schema evolution can be tracked across versions,
- analytics workflows remain reproducible,
- and semantic integration can be aligned with version-specific schemas.
- scalable delta computation (e.g., inspired by change data capture algorithms [Labio and Garcia-Molina, 1996], (see appendix B.2))

(5) Immediate metadata extraction at ingestion time: Effective metadata management begins at the point where data enters the system. SEDAR therefore performs automatic metadata extraction immediately after ingestion, ensuring that schemas, provenance information, structural characteristics, and preliminary statistical profiles are established consistently across datasets. This metadata serves multiple critical roles in later layers:

- enabling semantic alignment (RQ3),
- supporting dataset discovery and integration,
- tracking schema evolution,
- assisting non-technical users in understanding dataset structure (RQ4).

The metadata extraction component was designed to operate independently of the data's storage format. Distributed processing technologies demonstrated that structural metadata can be extracted efficiently even from large and complex datasets.

To operationalize these principles, SEDAR adopts a microservice architecture for its ingestion layer (figure 4.3). Rather than implementing ingestion as a monolithic module, the workflow is decomposed into conceptual services with clear responsibilities:

- an API service that governs user interaction and dataset definitions,
- an ingestion service responsible for execution logic,
- a continuation service that enforces continuous or scheduled ingestion lifecycles,
- a messaging layer mediating asynchronous coordination,
- and dedicated services for metadata extraction and statistical profiling.

The ingestion module allows additional code be executable during ingestion using so-called **plugins**, that can be attached to a data source and executed at predefined points. First, at data loading; plugins can take over the task of creating a DataFrame, replacing the standard function for loading data. Second, during post-loading; plugins can manipulate the DataFrame

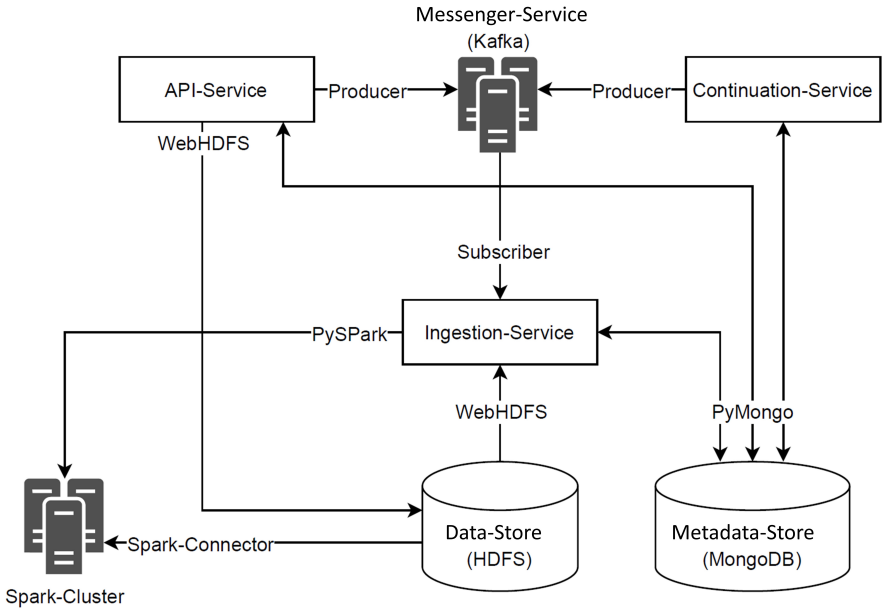


Figure 4.3: *SEDAR*'s *Ingestion Layer* as depicted by Martin [2022].

after loading. While this conflicts with a data lake's ELT principle of storing raw, unchanged data, such post-processing may be useful for cheap processing operations, e.g., data streams, where data is converted from byte blocks to a structured format.

Execution of an ingestion

The ingestion workflow in SEDAR follows a structured lifecycle that ensures heterogeneous data can be acquired, interpreted, versioned, and incorporated into the polystore consistently. Figure B.1 illustrates this process in a generalized flow.

At a high level, each ingestion begins with an initialization phase, during which the system retrieves the dataset definition, determines the ingestion mode (batch or stream), and resolves any configuration parameters required for the upcoming steps. Based on a dataset's declared read type, the system then decides whether the ingestion will be handled through a general-purpose processing engine or by a lightweight transfer operation for raw files.

A central feature of the workflow is the system's ability to incorporate extensibility through plugins. Before data is processed, optional plugins may be invoked to handle specialized access methods or custom parsing logic.

Once data is acquired, the workflow transitions into an interpretation phase, where the system determines whether the incoming dataset represents a new version or an update to an existing dataset. If differences must be computed, SEDAR supports change-data capture, allowing incremental updates to be recorded rather than re-ingesting entire datasets. This capability is particularly important for industrial contexts in which datasets evolve continuously over time and reproducibility of analyzes depends on maintaining precise historical states.

The storage phase assigns the ingested data to the appropriate backend within the polystore architecture. Conceptually, SEDAR distinguishes three `SAVE_TYPES`:

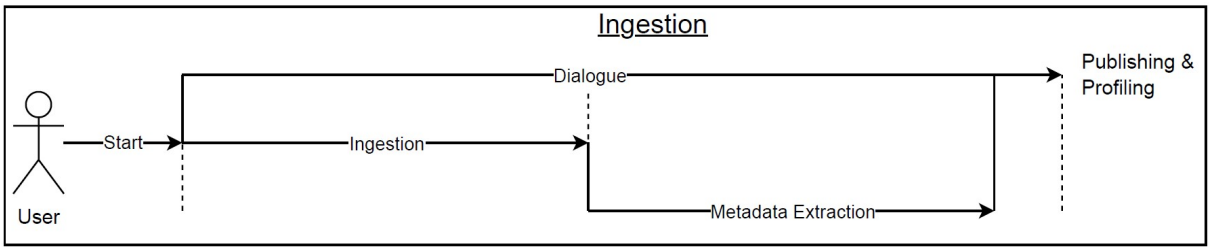


Figure 4.4: Data ingestion process.

- **Custom**: Users may decide to store the ingested data directly into one of the internal dedicated storage databases (see section 4.3.2), either PostgreSQL, Neo4j or MongoDB.
- **Delta**: Storage as DeltaTable (see section 4.3.2) in HDFS with versioning,
- **Default**: Storage as Parquet file in HDFS without versioning.

This decision is guided by dataset characteristics and system policies rather than by the ingestion logic itself, reflecting separation between logical ingestion workflows and physical data management.

For streaming sources or continuously updated datasets, the ingestion lifecycle concludes with an optional continuation phase, in which the system remains active and responsive to new data until a termination condition is met. This mechanism enables long-running monitoring pipelines in laboratory or production environments.

Conceptually, ingestion lifecycle follows the following workflow:
initialization $\rightarrow$ acquisition $\rightarrow$ optional transformation $\rightarrow$ versioning $\rightarrow$ storage $\rightarrow$ continuation.

Metadata Extraction

Metadata extraction constitutes a critical step in SEDAR's data-lifecycle management. Because datasets arriving in industrial environments are highly heterogeneous and may evolve over time, a data lake that merely stores raw files without contextual information quickly becomes unusable. **Metadata extraction is** therefore an **immediate and mandatory post-ingestion phase**, ensuring that every dataset is accompanied by a coherent structural description, provenance information, and semantic anchors that enable later integration and reuse.

Figure 4.4 provides a high-level view of this process. After ingestion has been triggered by the user, metadata extraction is automatically initiated in a dedicated service. This separation of responsibilities reflects the system's microservice design: ingestion focuses on acquiring data reliably, while metadata extraction interprets the acquired content and enriches it with descriptive knowledge. Only after both steps have been completed can a dataset be published within SEDAR and exposed to other users and analytical components.

Metadata extraction as an interpretation step: A central design principle in SEDAR is that the system does not rely on external metadata. Instead, all descriptive information is derived directly from the ingested artifact. This ensures robustness to data-source heterogeneity and avoids dependencies on external schema definitions that may be incomplete, inconsistent, or

unavailable.

During metadata extraction, the system inspects the internal structure of the dataset to determine its classification and corresponding extraction strategy. Figure 4.5 illustrates this decision flow. Conceptually, SEDAR distinguishes among three broad categories:

- **Unstructured data:** Files or binary artifacts without an inherent schema. These are analyzed using generic content-based metadata extraction methods [Quix et al., 2016], producing descriptors such as file type, encoding, size, and basic content signals.
- **Structured data:** Tabular datasets where attributes and types can be unambiguously identified. For such datasets, SEDAR derives a clean attribute-level schema and records structural metadata that supports downstream integration and query optimization.
- **Semi-structured data:** Hierarchical datasets recognized through file endings (.xml, .json, .csv), often represented as nested records or arrays. These require recursive inspection to reveal their internal structure. SEDAR extracts a logical schema describing objects, lists, and nested attributes, allowing them to participate in the system's unified representation model.

Supporting schema evolution and propagation of provenance and annotations: In industrial environments, datasets frequently evolve: attributes may appear, disappear, or change structure. SEDAR therefore incorporates schema versioning and change detection as first-class concepts. When a new version of an already-ingested dataset is detected, the system:

- compares the new structural description with previous versions,
- identifies added, removed, or modified attributes, and
- records the resulting schema as a new version linked to its predecessors.

This ability to track schema evolution ensures historical analyses remain reproducible while enabling users to understand how data sources change over time—an essential requirement for data governance in industrial settings.

If datasets carry provenance information (e.g., derivation chains or parent datasets), SEDAR transfers the relevant annotations to the new version whenever this is unambiguous. This mechanism supports semantic continuity across dataset versions and allows downstream tasks, such as integration, linking, and analytics, to operate on consistent metadata even as the underlying data evolves.

Profiling of Source Data

The final step after the ingestion and the metadata extraction is the data profiling of the source data, which is performed to automatically obtain initial statistical and analysis-relevant information performed independently in a standalone microservice. For this purpose, ***Deequ***, an open source library for data quality management with Apache Spark developed by Amazon Research[1] is integrated. Its main goal is to enable "unit tests for data" [Schelter et al., 2019], thereby automating data quality verification on large datasets. The profiling process derives information for each column such as the data type, a completeness metric or the statistical

[1] https://github.com/awslabs/deequ, accessed on 02.05.2026

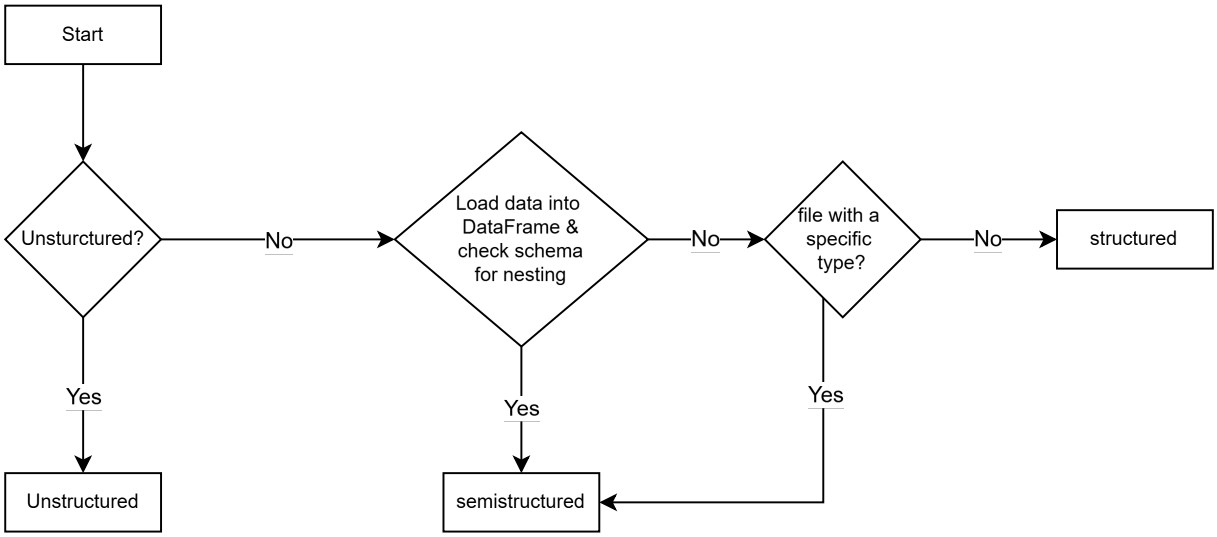


Figure 4.5: Metadata extraction process

figures in case of numerical data.

One challenge during development was that Deequ is fundamentally not applicable to nested data. However, this issue is circumvented by ***flattening***[2] nested data beforehand. Here, flattening means converting a nested data structure into a relational one, by rolling out all arrays and nested fields into columns of a "flat" table. This way Deequ can also be used for nested data, although there may be cases where the profiling results do not produce meaningful outcomes.

4.3.2 Storage Layer

The Storage Layer provides SEDAR's foundation for managing heterogeneous datasets in a scalable and lifecycle-aware manner. It combines five architectural principles implemented to comprehensive metadata management (section 4.3.2):

Key principles

(1) A unified repository for raw and versioned data SEDAR adopts a distributed file system as its default storage for ingested artifacts. This substrate provides a neutral, format-agnostic environment that can host:

- unstructured files in their original form,
- structured or semi-structured datasets in columnar formats, and
- versioned datasets as logical tables with temporal reconstruction.

To support dataset evolution and reproducibility, SEDAR integrates a transactional table format (*DeltaTables*) enabling ACID-aware updates and time-travel queries [Armbrust et al., 2020]. Versioning is treated as an architectural feature: evolving datasets maintain a log of structural and content changes, allowing analytical workflows to retrieve historical states.

[2] https://gist.github.com/nmukerje/e65cde41be85470e4b8dfd9a2d6aed50, accessed on 02.05.2026

(2) Organizing stored data within Workspaces SEDAR organizes stored data inside a Workspace, a logical container grouping datasets, metadata, and user-defined workflows. Workspaces serve two purposes:

- collaboration and access control for shared work;
- logical isolation of datasets, plugins, and derived assets.

Each Workspace contains:

- source files preserving original artifacts,
- plugins encapsulating ingestion logic,
- data folders for structured, semi-structured, unstructured, or versioned datasets.

This structure ensures lifecycle traceability and simplifies metadata management by mapping ingestion events to stored artifacts.

(3) Polystore architecture for heterogeneous data models A central contribution of SEDAR is its polystore design, integrating multiple database engines under a unified access layer. Rather than enforcing a single model, SEDAR leverages:

- relational storage for structured tabular data,
- document-oriented storage for nested or schema-flexible data,
- graph storage for interconnected entities and relationship-centric analyses.

The polystore layer supports:

- model-based data placement,
- scalable performance via engine-specific workloads,
- unified access through a consistent interface.

(4) Semantic storage for Knowledge Graphs and ontologies To support Semantic Data Management (see chapter 5), SEDAR incorporates a dedicated RDF triple store. This component enables:

- ontology and knowledge-graph storage,
- semantic reasoning,
- integration of metadata with formal concepts.

(5) Search and index management The Storage Layer includes a full-text and attribute-level indexing service that allows users to:

- search across heterogeneous datasets,
- query nested structures,
- discover datasets by content,
- explore data without prior schema knowledge.

Indexing is optional and can be enabled during ingestion. When indexed datasets are updated, SEDAR automatically synchronizes the index, ensuring lifecycle consistency. This capability addresses limited dataset discoverability in industrial data lakes.

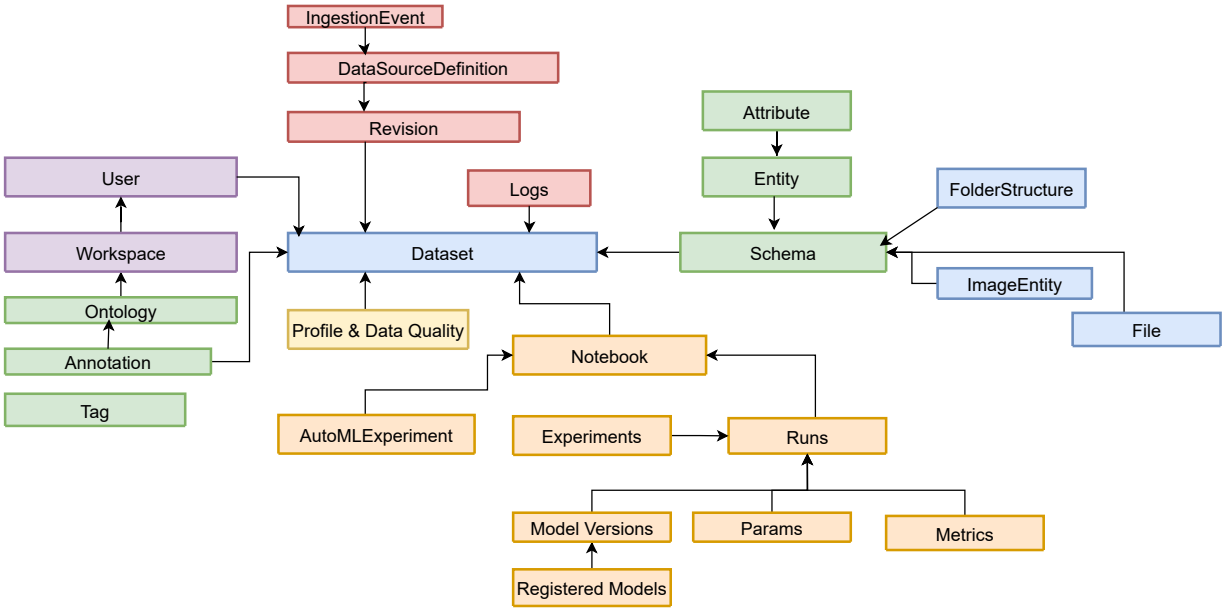


Figure 4.6: SEDAR's metadata model, colour-coded by architectural concern: semantic layer (green), data assets/representations (blue), lifecycle & provenance (red), governance context (purple), quality signals (yellow), and analytics/model lineage (orange).

Metadata Model for Data Cataloguing

Metadata forms the structural backbone of SEDAR. The consolidated metadata model shown in figure 4.6) integrates ingestion metadata, dataset cataloguing, semantic information, analytical artifacts, and usage logs into a unified structure. Every subsystem, such as ingestion, storage, semantic enrichment, workflow execution, and analytics, relies on a unified metadata framework that captures what data exists, where it originates, how it evolves, and how it should be interpreted. Accordingly, SEDAR adopts a ***metadata-first*** design principle: no dataset enters the system without a corresponding, multi-level metadata representation. This ensures discoverability, version traceability, semantic interoperability, and robust governance. The architectural design employs a hybrid storage strategy conceptually composed of: (1) a **graph-oriented metadata layer** for relationships (schemas, lineage, ontology links, access control), (2) a **document-oriented metadata layer** for flexible structures (ingestion configurations, nested properties, logs), and (3) a **relational metadata layer** for capturing analytics.

Although a fully consolidated metadata store would be conceptually desirable and could simplify the overall system architecture, realizing such a unification would have required extensive redevelopment effort beyond the scope of this work. SEDAR evolved incrementally, and its metadata subsystems rely on storage technologies that are already well aligned with their respective requirements: Neo4j for highly connected semantic and lineage metadata, MongoDB for flexible and rapidly evolving ingestion metadata, and a relational backend inherited from MLflow. Replacing or re-engineering these established components would have demanded substantial time and resources and would have shifted the focus away from the architectural and methodological contributions of this dissertation. The resulting hybrid design therefore

reflects a pragmatic polyglot persistence approach:

DataSourceDefinitions represent the system's abstract knowledge of an external source. It captures human-readable descriptors, identifiers, ingestion settings, and any associated semantic or structural expectations. Rather than overwriting changes, evolving configurations are captured through revisioning.

Revisions are created by each modification to a DatasourceDefinition. Revisions isolate ingestion behavior in time: analyzes, workflows, and lineage relationships can always reference the configuration used at a particular moment. This supports reproducibility and metadata-temporal reasoning.

Ingestion Events encode individual execution runs, their state transitions, timings, and error conditions. By linking events to revisions, SEDAR reconstructs not only the data but also its operational history. This provides full transparency over how data entered the system, which revision was applied, and whether errors occurred.

Datasets as the central metadata abstraction aggregate descriptive metadata such as title, description, and ownership information; schema and structural metadata; lifecycle information including creation time, updates, and version history; access control settings; semantic annotations and tags; lineage relationships capturing parent and child datasets; polymorphism links; and references to associated notebooks or analytical artefacts. Thus, a dataset in SEDAR is a semantically rich, versioned object not merely a pointer to stored data.

Workspaces facilitate governance and organizational context by grouping datasets, ontologies, tags, users, and analytical artifacts into logical collaboration units. They serve as containers for governance boundaries, access control rules, semantic scoping—specifically the ontologies available within a Workspace—and shared analytical resources. The metadata model encodes read/write/delete permissions directly on metadata objects, enabling fine-grained governance.

Semantic enrichment: Ontologies, Annotations, and Tags. Ontologies imported into a Workspace enable semantic annotations at dataset, entity, or attribute level. These annotations produce *Tags*, which support semantic search, dataset grouping by shared concepts, similarity detection, and the linking of datasets across versions and representations. Through this semantic metadata, the system is transformed from a conventional data lake into a semantic data fabric.

Entities and Attributes. For structured or semi-structured data, *Entities* model logical tables or nested structures, while attributes capture data type and nullability, nesting and hierarchical structure, primary or foreign key roles, semantic annotations, and attribute-level statistics. These structures support schema evolution, matching, profiling, and annotation workflows. Un-

structured datasets are represented by *File* metadata describing extracted properties and file characteristics.

Usage Logging and Lineage. A versioned *Log* model records all interactions, including creation, reads, updates, and deletions. Dataset lineage is explicitly represented, enabling the reconstruction of processing histories and supporting reproducibility.

4.3.3 Transformation Layer

Data transformation is a central capability of SEDAR, enabling users to refine, combine, and derive new data products from ingested datasets. The platform provides three complementary mechanisms that differ in their level of abstraction, required technical expertise, and integration depth: (1) programmatic transformations in Jupyter notebooks, (2) SQL-based transformations in the *Query* tab, and (3) graphical workflow construction in the *Workflow Module*. Together, these approaches support both expert users who require maximal flexibility and non-technical users who benefit from guided, visual tooling.

Jupyter-Based Transformations The tight integration of Jupyter notebooks offers the most flexible transformation environment. Datasets can be loaded as `DataFrame` objects and processed with arbitrary Python or PySpark code. By leveraging the *SEDAR-API* (see section 4.3.4), transformed datasets can be written back to the platform with minimal boilerplate. This approach is suited for complex analytical workflows and machine learning pipelines, where users require full control over data wrangling, feature engineering, or domain-specific analytical transformations. It also serves as the reference mechanism for validating the correctness of transformations executed via more abstract layers.

SQL-Based Transformations via the Query Tab For users who prefer a declarative interface, the *Query* tab enables SQL-like transformations directly within the dataset view. Queries are executed using SparkSQL, allowing the system to apply a uniform query language across heterogeneous storage systems. Results can be previewed, and uploaded as a new *DataSet*. This mechanism supports lightweight transformations such as projection, filtering, or simple joins, and is particularly valuable in early-stage data exploration.

The Workflow Module provides a higher-level transformation interface that abstracts away code entirely. It implements a flow-based programming paradigm, where users construct workflows by connecting modular nodes representing data operations. This design lowers technical barriers while remaining expressive enough for multistep transformations.

From a backend perspective, workflows are represented as nested JSON structures describing a directed acyclic graph of operations. Nodes correspond to PySpark-based operators, such as `JOIN`, `FILTER`, `SELECT`, `DISTINCT`, `GROUP BY` operators with aggregations `COUNT`, `AVG`,

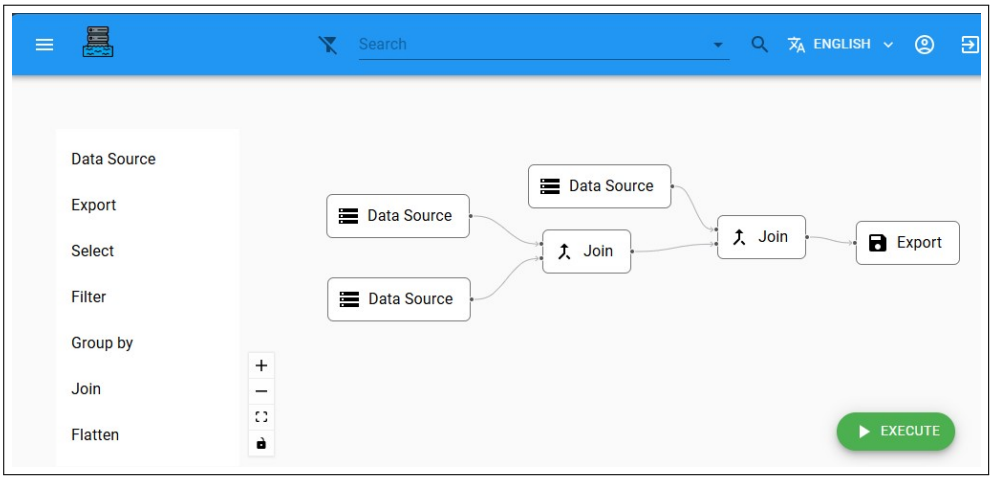


Figure 4.7: Example of a data processing workflow defined in the *Workflow Module*.

`SUM`, `MAX`, `MIN`), and the `flatten` transformation for nested structures.

The graphical interface is implemented using *ReactFlow*, enabling users to construct pipelines interactively. The corresponding `ViewModel` translates the graph into a structured representation that the backend can execute deterministically. The execution engine resolves the workflow recursively: leaf nodes define the initial *DataSets*, while parent nodes apply the designated transformations and pass results upward. The final output is a Spark DataFrame that can be stored as a new versioned dataset with provenance captured automatically.

Figure 4.7 illustrates a representative workflow, demonstrating how end users can visually construct transformation pipelines.

Data Cleaning Pipelines To support data quality management, SEDAR integrates profiling, constraint validation, and data-cleaning pipelines based on *Deequ*. These capabilities complement transformation workflows by ensuring that derived datasets meet domain-specific quality thresholds.

The profiling process computes completeness, uniqueness, basic descriptive statistics, and type distributions, which are cached in the metadata model for efficiency. Based on these profiles, the constraint suggestion service proposes validation rules such as `Completeness`, `Uniqueness`, or `ValueRange` constraints. Users may accept suggestions or define constraints manually.

Validation results produce quantitative quality indicators for each constraint. These indicators form the basis for deriving cleaning filters, which can then be executed to produce cleaned datasets. All operations—including constraint choices, validation results, and cleaning actions—are recorded in the metadata model to ensure transparency and reproducibility.

A dedicated frontend interface exposes these capabilities within the *Profile* tab of a dataset, allowing users to inspect profiles, apply constraints, review validation outcomes, and trigger cleaning workflows. Unlike transformation workflows, which focus on structural and relational operations, the cleaning pipelines target attribute-level quality improvements grounded in formal

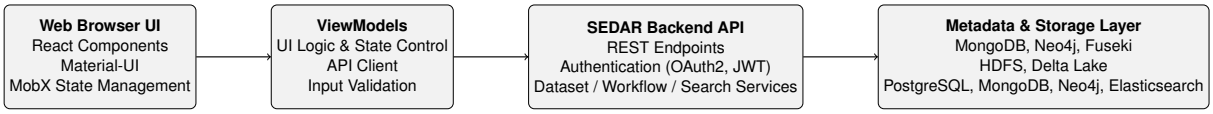


Figure 4.8: Conceptual architecture of the Interaction Layer in SEDAR (left-to-right).

data-quality rules.

4.3.4 Interaction Layer

The *Interaction Layer* provides all user-facing access points to SEDAR. Whereas the lower layers govern ingestion, storage, and metadata management, the interaction layer exposes these capabilities to end users through three complementary interfaces:

1. a **web application** for visual, interactive exploration,
2. a **Python API** for programmatic integration (see section 4.3.4), and
3. a **natural-language interface** for conversational access (see chapter 7).

This chapter introduces the conceptual structure of the web application and its major components. Because the system includes numerous user-interface screens, only conceptual overviews are presented here; detailed screenshots and UI walkthroughs are provided in the Appendix (see appendix B.3).

Web Application Architecture

The web application is a single-page application implemented in *React*. Its architecture follows a component-based design and separates presentation from interaction logic:

- the **MainComponent** of each view defines layout, markup, and styling, and
- the corresponding **ViewModel** encapsulates business logic and bidirectional communication with the backend.

Stateful interactions are handled using *MobX*, which maintains observable state and ensures predictable UI updates as metadata or dataset views change. To accelerate development and enforce design consistency, the interface employs *Material UI* components.

Figure 4.8 illustrates the high-level structure of the interaction layer and its coupling with backend services.[3]

Authentication User authentication is performed using both traditional credential-based login and OAuth 2.0 [Hardt, 2012]. For the production environment, OAuth is backed by a dedicated GitLab instance, enabling Single Sign-On across systems such as *SEDAR* and *JupyterHub*. After successful authentication, the server issues a signed JSON Web Token (JWT), which is stored client-side and included in subsequent requests. User sessions therefore remain stateless and horizontally scalable across backend instances.

[3] Screenshots of individual interfaces (login, dashboard, ingestion dialog, dataset view, etc.) are provided in the Appendix.

Dashboard After authentication, users are redirected to the *Dashboard*, which serves as the landing page for each *Workspace*. The dashboard provides:

- quick access to the user's favorite datasets,
- monitoring of component health (for administrators),
- links to data ingestion, semantic management, workflows, and machine-learning modules.

Screenshots and UI examples are available in appendix B.3.

Workspace Management provides a governance layer for grouping datasets, assigning ontologies, and configuring access rights. The owner of a workspace can:

- add or remove users,
- assign read, write, and delete permissions,
- modify metadata such as title and description,
- inspect workspace-level statistics.

In the backend, these actions translate directly into modifications of the metadata graph and corresponding access-control edges (see section 4.3.2).

Ingestion Interface guides users through the creation of a *DatasourceDefinition*, upload of source data and plugins, and specification of metadata needed to publish the resulting *Dataset*. The interface consists of four conceptual stages:

1. **Definition stage:** A user provides the ingestion specification. Templates for CSV, JSON, JDBC sources, streams, and custom definitions are available.
2. **Metadata stage:** Users enter descriptive metadata (title, description, tags, temporal coverage, authorship, licence). Provenance suggestions are autofilled when parents exist.
3. **Schema stage:** Extracted schemas and classifications (structured, semi-structured, unstructured) are presented. Users may override inferred data types.
4. **Publishing stage:** Users review their selections, choose visibility settings, and (optionally) enable indexing.

The ingestion UI is intentionally generic: all ingestion logic remains server-side, while the client provides guided configuration. Examples of each step are shown in the Appendix (appendix B.3).

The Dataset View & Search serve as the primary interface for interacting with ingested and transformed data. They are organized into functional tabs that expose metadata stored in the underlying model:

- *General Tab*: Displays descriptive metadata, schema summaries, linked datasets, polymorphic siblings, and basic statistics.
- *Profile Tab*: Visualizes the schema using a graph-based view (via *neovis.js*). For unstructured datasets, file properties extracted via Apache Tika are shown. Structured datasets display Deequ-based statistics and attribute-level metadata.

```
from sedarapi import SedarAPI
sedar = SedarAPI("http://127.0.0.1:5000")
admin = sedar.login("user@example.com", "pw123")
ws = sedar.create_workspace(title="SEDAR-API")
```

Listing 4.1: Creating a workspace using the SEDAR-API

- *Query Tab*: Provides a visual query builder based on the dataset's schema. Queries are converted to SparkSQL and executed server-side, with results optionally saved as new datasets.
- *History Tab*: Shows previous versions, associated *DatasourceDefinitions*, and—when applicable—computed diffs between DeltaTable versions.
- *Lineage Tab*: Visualizes provenance as a hierarchical tree of parent and derived datasets, including workflow-generated datasets. Schema comparison dialogs allow inspecting structural changes between related datasets.
- *Analysis Tab*: Manages Jupyter, MLflow, or AutoML notebooks associated with the dataset. Users may create notebooks tied either to a specific version or to the latest version.
- *Options Tab*: Allows dataset owners to configure access rights, trigger re-indexing, or delete datasets.
- *Search Interface*: Supports full-text, semantic, attribute-level, and metadata-based search across all workspaces. Depending on the search target, queries are executed either:
  - in Neo4j (semantic search, ontology-based matching, relationship traversal), or
  - in Elasticsearch (text search, keyword filters, range queries).

  Filtering dimensions include owner, schema type, tags, temporal ranges, row counts, file size, and notebook-related metadata. Sorting and pagination ensure efficient navigation across large result sets.

The SEDAR-API

While the web interface is suitable for interactive work, programmatic access requires a simplified abstraction over the backend's heterogeneous REST API. To this end, an external Python library called the *SEDAR-API* was developed [Kuth, 2023]. It provides:

- a unified access point (`SedarAPI` class),
- consistent CRUD abstractions for all major resources,
- automatic session management and authentication,
- standardized return types and error handling,
- idiomatic Python bindings for datasets, workspaces, ontologies, tags, notebooks, and ML experiments.

The library follows the *Facade* design pattern, hiding the complexity of dozens of backend endpoints behind a few intuitive methods. Internally, the library delegates requests to a helper module that manages HTTP communication, logging, sessions, and serialization. An example is shown in listing 4.1.

4.4 Conclusion

This chapter addressed RQ2 by presenting the architectural principles, design decisions, and functional components that constitute the SEDAR platform. The system was developed in response to the complex technical and organizational requirements of industrial chemistry environments, where heterogeneous data sources, evolving schemas, and domain-specific knowledge coexist with the need for reproducibility, traceability, and scalable processing.

SEDAR contributes a **unified architectural approach** that integrates (i) a format-agnostic ingestion pipeline, (ii) a polystore-based storage and access abstraction, (iii) metadata- and provenance-centric data governance, and (iv) semantic enrichment mechanisms that embed domain knowledge directly into the data lifecycle. While each of these elements is grounded in existing research, their consolidation into a coherent microservice-based data-management platform represents a novel contribution to research on data architectures in industrial settings.

The system's metadata-driven and ontology-aware design provides the foundation for answering RQ3. By tightly integrating semantic models into ingestion, storage, and interaction workflows, SEDAR enables the consistent interpretation, linking, and reuse of heterogeneous datasets. This semantic layer enhances interoperability and supports domain experts in navigating complex data landscapes without requiring deep technical expertise.

Finally, the interaction mechanisms presented in this chapter ranging from the graphical web interface to the programmatic *SEDAR-API* address RQ4. These components demonstrate how a data-management platform can lower cognitive and technical barriers for non-technical users while still supporting advanced analytical workflows. The integration with notebook environments and workflow orchestration further illustrates how SEDAR supports iterative, user-driven data exploration and machine-learning development.

Taken together, the concepts and mechanisms introduced in this chapter provide a research-driven blueprint for data-management platforms in industrial contexts. In section 8.7 we evaluate the research contributions and the assess the architecture with DATER framework.

5

Managing Semantics in SEDAR

In the next wave of AI research, if we want to make more helpful and useful machines, we've got to bring back the contextual understanding.

Fei-Fei Li

Semantic data management is fundamentally about making data interpretable, integrable, and *accessible* in terms of its domain meaning rather than its raw technical representations. In response to RQ3, we show how semantic representations (e.g., vocabularies, labels, and mappings) can be derived from ingested artifacts and operationalized to support cross-source discovery, integration, and access in SEDAR.

While semantic approaches are conceptually powerful, their practical adoption is still hindered by high modeling costs and a lack of automation. A central contribution of this chapter is therefore to examine how automated methods, including LLMs, can support individual tasks. Building on these insights, we show how SEDAR integrates semantic technologies into its metadata, ingestion, and access layers to support scalable, context-aware data management.

In particular, we highlight how semantic models created and refined within SEDAR can be systematically transformed into *ontology–to–source mappings*, enabling an access layer that virtualizes heterogeneous data through a unified conceptual interface with a RML mappings and a technique called ontology-based data access (OBDA).

This chapter is largely based on the publications [Hoseini et al., 2023, 2024d,b,a].

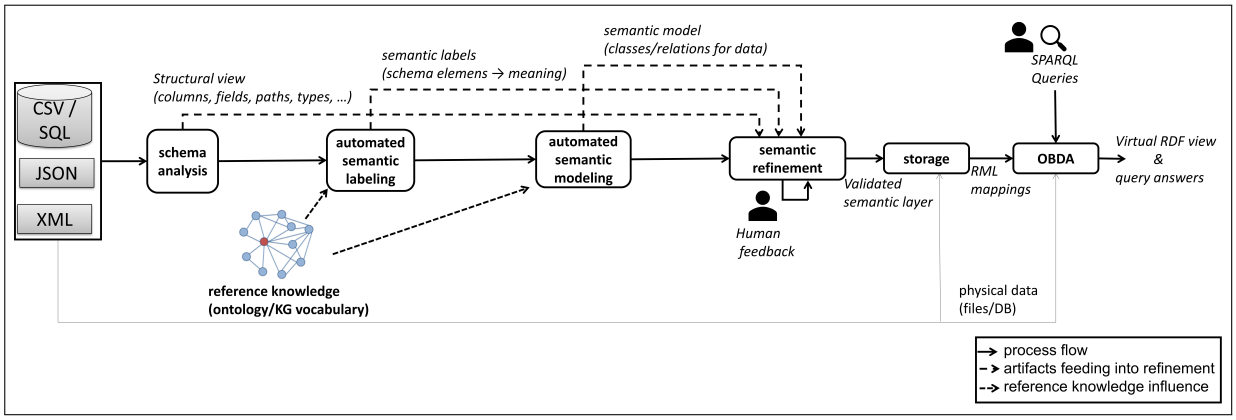


Figure 5.1: Phases to generate semantic models for OBDA [Paulus et al., 2021b, Hoseini et al., 2024d]

As illustrated in figure 5.1, this chapter is structured along semantic layer extraction for heterogeneous sources operationalized as RML mappings to enable OBDA.

Chapter Outline

This chapter is structured as follows:

- Section 5.1 – Motivation and challenges of semantic data management in heterogeneous industrial environments.
- Section 5.2 – Conceptual foundations of ontology-based data management and semantic modeling.
- Section 5.3 – Automated semantic labeling: LLM-based methodology.
- Section 5.4 – Automated semantic modeling: Conceptual role of LLMs for semantic relation inference and model creation.
- Section 5.5 – Semantic refinement, with a focus on the conceptual workflow implemented in PLASMA.
- Section 5.6 – Conceptual foundations of storage and ontology-based data access for unified querying across heterogeneous sources.
- Section 5.7 – pipeline implementation in SEDAR.
- Section 5.8 — concludes the chapter by assessing the contribution with regards to the research question (section 1.2).

5.1 Introduction

Data sources that are semantically well-annotated can be identified and interpreted by leveraging conceptual representations of the data and by comprehending the provided context information stored in a semantic model. However, a huge initial overhead, coming from the time-consuming manual process of creating meaningful semantic descriptions for data sources, hampers the widespread adoption of semantic data management (SDM) in practice [Hoseini

et al., 2024d]. Creating semantic models entails deciphering the existing data source, consulting appropriate conceptualizations, and establishing connections between data attributes and concepts provided by the conceptualization. We conceptualize this process in five steps:

(1) Schema Analysis The process begins with identifying the structural properties of a dataset: attributes, nested elements, and data types. This step provides the syntactic foundation upon which semantic interpretation is built.

(2) Automated Semantic Labeling Automated semantic labeling, also known as *Semantic Type Detection* [Hulsebos et al., 2019], assigns ontology concepts or classes to individual attributes and is associated to *Semantic Table Interpretation* [Liu et al., 2022]. This requires learning correspondences between surface-level attribute representations and conceptual entities. As prior work shows [Vu et al., 2019a, Taheriyan et al., 2015], errors at this stage propagate through the pipeline, making high-quality labeling essential for downstream automation. Burgdorf et al. [2022] classify methods into

1. *schema-driven* [Pinkel et al., 2017, Paulus et al., 2018],
2. *data-driven* [Ramnandan et al., 2015, Pham et al., 2016, Abdelmageed and Schindler, 2021], and
3. *metadata-driven* [Deng et al., 2022].

Liu et al. [2022] additionally distinguish ML-based, heuristic, and feature-engineering approaches.

(3) Automated Semantic Modeling Building on attribute-level labels, the semantic modeling phase infers additional structure: relationships, constraints, and linking concepts. This step, also referred to as *semantic relation inference* [Futia et al., 2020], aims to construct a coherent, ontology-aligned semantic model of the dataset.

(4) Semantic Refinement Fully automated approaches remain imperfect. A human modeler therefore performs semantic refinement to correct mislabeled attributes, adjust relations, and ensure conceptual validity before the semantic model is finalized and stored.

(5) Ontology-Based Data Access (OBDA) OBDA is commonly considered a specific realization within the broader field of OBDM. In OBDA, so-called mappings connect data attributes to ontology concepts and form the technical basis for query rewriting. While mappings have formal semantics and stricter correctness conditions, conceptually they serve a similar role as semantic labels: linking physical data structures to conceptual entities. The semantic labels and semantic models introduced in this section define a technology-independent interpretation layer over heterogeneous data sources. In SEDAR, these abstractions are operationalized using OBDA, where semantic labels are transformed into executable RML mappings and queried using SPARQL. While the present section focuses on the conceptual foundations, the concrete realization of OBDA in SEDAR is discussed in detail in section 5.7.4.

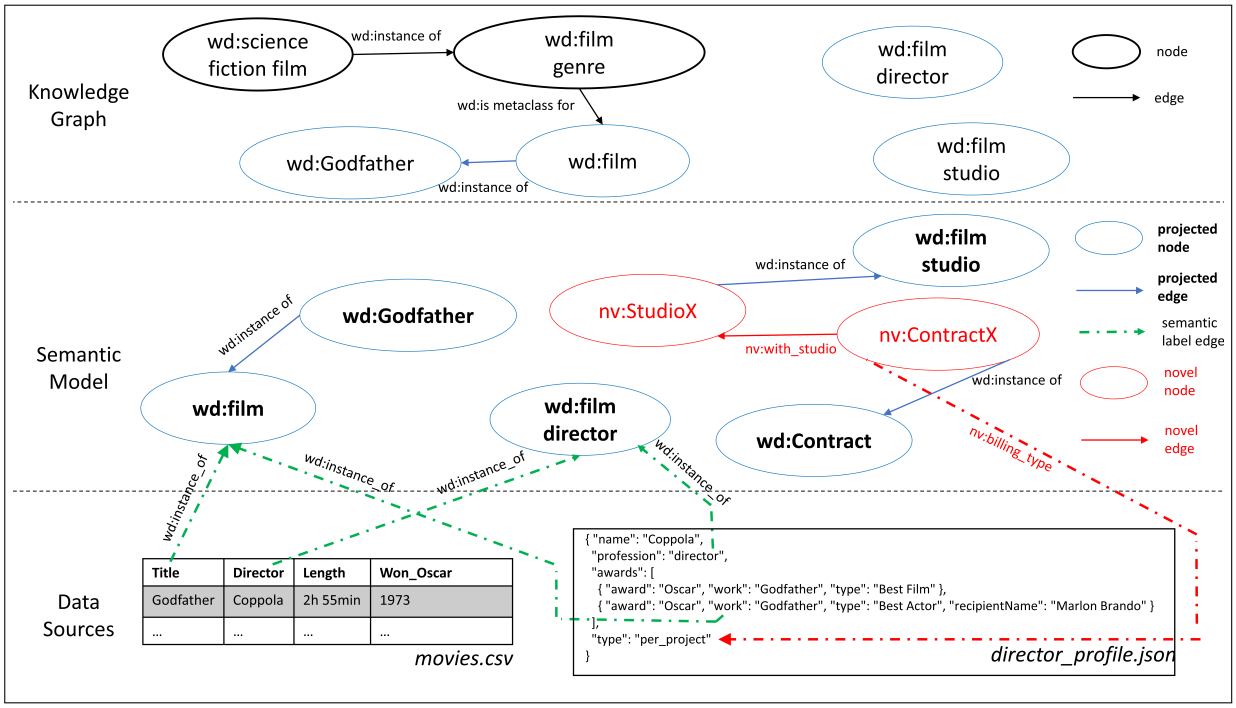


Figure 5.2: A semantic model for heterogeneous data sources. Most concepts of the semantic model are projected (blue) from the KG (black), but to describe specific data sets one may define also novel elements (red).

Mami et al. [2016] introduce the notion of semantic lifting: transforming raw data into a semantically annotated representation using labels (and optionally entire semantic models). Such transformations resemble ETL pipelines and may be expensive for large datasets, especially where freshness is required. Before turning to automation techniques and their integration into SEDAR, the next sections introduce theoretical foundations of *ontology-based data management* (section 5.2), describe contemporary approaches of automation (sections 5.3 and 5.4) as well as GUI-assisted creation of semantic models (section 5.5) and *ontology-based data access* (section 5.6). These concepts provide the formal machinery that underpins the entire semantic modeling pipeline: ontologies supply the conceptual vocabulary used for semantic labeling and modeling, while OBDA defines how semantic models can operationalize heterogeneous datasets through a unified, ontology-driven access layer.

5.2 Ontology-based Data Management

Despite substantial progress in Linked Data and Semantic Web research, Semantic Web technologies still operate at a smaller scale than modern big-data infrastructures. Achieving semantic interoperability at industrial scale therefore requires bridging expressive knowledge representations with large, heterogeneous datasets. A semantically enriched data source can be discovered using the conceptual representation provided by a so-called **semantic model**, and interpreted through the contextual information it encapsulates. Such models enable both more

expressive data management and heterogeneous data integration. Paulus et al. [2021a] characterize this field of ontology-based data management (OBDM) through two central processes:

1. the development and maintenance of conceptualizations (i.e., ontologies or in practice often entire KGs), and
2. the semantic enrichment of data sources by linking schema elements to these conceptualizations.

As organizations increasingly recognize the strategic value of data, such models are becoming standard components of data governance and master data management [Dibowski and Schmid, 2021, Hoseini et al., 2024d]. Although conceptualizations may be created from scratch, reusing existing ontologies is preferable due to the expertise-intensive nature of ontology engineering [Trajanoska et al., 2023, Meyer et al., 2023b].

The creation of semantic models involves analyzing the data source, identifying relevant KGs, and mapping data attributes to conceptual entities. This positions the KG as an overarching semantic backbone for an organization's data assets [Dibowski et al., 2020]. The resulting semantic model acts as a projection layer, connecting raw data with the knowledge layer [Pomp et al., 2018].

As illustrated in Figure 5.2, raw data in diverse formats (e.g., data sources such as tables, JSON) may share overlapping semantics. The semantic model, which includes the semantic labels, is expressed as a graph that reuses entities and relations from the KG (blue nodes & edges), and augments them as needed to capture dataset-specific semantics. Since incoming datasets may introduce new concepts, the semantic model and the underlying KG must evolve dynamically [Pomp et al., 2019a] (red nodes & edges).

5.2.1 Semantic Labels

Semantic labels connect elements of one or more datasets to a conceptualization (e.g., Wikidata) in order to make the intended meaning of the data explicit. This principle is independent of the concrete storage format: tables, JSON, and XML are treated uniformly as collections of objects from which key–value observations (including nested paths) are extracted.

The formalization in this section is deliberately chosen to be compatible with RML/R2RML. RML and its predecessor R2RML deliberately avoid introducing a unified formal data model for heterogeneous sources. Instead, they rely on the native query or selection languages of each format (e.g., SQL, JSONPath, XPath) to identify relevant data fragments. This design effectively delegates structural interpretation to the underlying data system and treats source access abstractly in terms of iterated objects and references. In the following, we adopt a similar abstraction relating the definitions back to figure 5.2 wherever applicable. Rather than formalizing relational schemas, JSON trees, or XML documents individually, we model each source through a set of objects and an extraction function that yields key–value observations. This captures the information required for semantic labeling while remaining compatible with the access patterns used in RML mappings. This alignment is made explicit in section 5.2.2.

Source s	$k \in \mathcal{K}_s$ **(selector)**	$v \in \mathcal{V}$ **(value)**
CSV	`Title`	`Godfather`
JSON	`$.awards[1].type`	`Best Actor`
XML	`/person/name/text()`	`Coppola`

Table 5.1: Examples of extracted key–value observations across source formats from the running example in figure 5.2.

Data model

Let S be a set of data sources (datasets), e.g., individual CSV files, JSON documents or XML files. For each source $s \in S$, let

- $\mathcal{R}_s$ be a set of object identifiers (e.g., table rows selected by an SQL statement, JSON objects selected by a JSONPath iterator, XML nodes selected by XPath), and
- D_s be an extraction function which maps each object $r \in \mathcal{R}_s$ to a finite set of key–value observations:

$$D_s(r) \subseteq \mathcal{K}_s \times \mathcal{V},$$

where $\mathcal{K}_s$ is a set of selectors (attribute names, JSONPaths, XPaths) and $\mathcal{V}$ is a set of observed values (strings, numbers, dates, nested objects, lists, ...). In each source, the object identifier r identifies the unit of abstraction (e.g., a table row or a JSON object) and anchors all extracted values to a single logical entity. The selector k identifies the structural role of a value within that object. The notation is exemplified in table 5.1.

This abstraction closely mirrors the design used in RML mappings. In particular, Objects $r \in \mathcal{R}_s$ correspond to *logical source iterations*, i.e., the elements produced by an RML iterator (such as table rows selected by an SQL query, JSON objects selected by a JSONPath iterator, or XML nodes selected by XPath) and selectors $k \in \mathcal{K}_s$ correspond to *references* in RML, which specify how values are accessed within each iterated object (e.g., column names, JSONPath expressions, or XPath expressions). The extraction function D_s corresponds to the *evaluation of references* for a given logical source object, yielding the set of observed key–value pairs.

We define the set of data elements as a disjoint union of schema-level and instance-level elements:

$$\begin{aligned}
\mathcal{C}_{\text{schema}} &= \{(s,k,\star) \mid s \in S,\, k \in \mathcal{K}_s\}, \\
\mathcal{C}_{\text{instance}} &= \{(s,k,v) \mid s \in S,\, k \in \mathcal{K}_s,\, \exists r \in \mathcal{R}_s :\ (k,v) \in D_s(r)\} \quad \cup \\
&\qquad \{(s,r,k,v) \mid s \in S,\, r \in \mathcal{R}_s,\ (k,v) \in D_s(r)\}, \\
\mathcal{C} &= \mathcal{C}_{\text{schema}} \cup \mathcal{C}_{\text{instance}}.
\end{aligned}$$

Schema-level elements $(s, k, \star)$ refer to selector k in source s in general. Instance-level elements refer to observed values, either object-abstracted (s, k, v) or object-specific (s, r, k, v). To distinguish schema-level from instance-level elements (and to avoid collisions between identical keys in different sources), we use source-qualified data elements:

- **Schema-level element:** $(s, k, \star)$ meaning *"selector k in source s in general, applying to all occurrences."*
- **Instance-level element:** either (s, k, v) meaning *"the value v under selector k in source s"* (abstracted across objects), or (s, r, k, v) meaning *"the value v under selector k in the specific object $r \in \mathcal{R}_s$."*

For the schema level, we have for example

$$(s_{\text{csv}}, \texttt{Director}, \star) \in \mathcal{C}_{\text{schema}} \quad \text{and} \quad (s_{\text{json}}, \texttt{\$.profession}, \star) \in \mathcal{C}_{\text{schema}}.$$

For the instance level (object-abstracted), we have for example

$$(s_{\text{csv}}, \texttt{Director}, \texttt{Coppola}) \in \mathcal{C}_{\text{instance}} \quad \text{and} \quad (s_{\text{json}}, \texttt{\$.profession}, \texttt{director}) \in \mathcal{C}_{\text{instance}}.$$

Here $\star$ denotes "all values / the key as such." This renders "column" a special case of "selector" and "cell value" a special case of "value occurrence." For brevity, we subsequently omit the explicit source qualifier s and, unless an individual occurrence must be identified, also the object identifier r. We then write $(k, \star)$ for schema-level elements, (k, v) for value-level elements (across objects), and (r, k, v) when referring to a specific object occurrence, assuming that k implicitly ranges over source-qualified selectors (or that the relevant source is clear from context). Semi-structured sources allow different, equally valid choices of object granularity.

When a source contains repeated or ordered fields (arrays, repeated XML nodes), selectors $k \in \mathcal{K}_s$ include positional information (e.g., JSONPath indices, XPath position predicates), so $D_s(r)$ remains finite while order can be recovered from k. Note that `$.awards[*].type` is a schema-level selector pattern, whereas elements of $D_s(r)$ use concrete selectors with positional information, e.g., `$.awards[0].type` and `$.awards[1].type`. For the JSON field `"recipientName":"Marlon Brando"` (occurring in the second award entry), the same observation can be represented in equivalent ways; we illustrate two common choices:

(1) Root-as-object: $\mathcal{R}_s = \{r_{\text{root}}\}$, encode the occurrence in k (in JSON, $r_{\text{root}} = \$$):

$$(s, r_{\text{root}}, \texttt{\$.awards[1].recipientName}, \texttt{"Marlon Brando"}).$$

(2) Awards-as-object: $\mathcal{R}_s = \{\, r \mid r \text{ selected by } \texttt{\$.awards[*]} \,\}$, encode the occurrence in r:

$$(s, \texttt{\$.awards[1]}, \texttt{"recipientName"}, \texttt{"Marlon Brando"}).$$

For our running example in Figure 5.2 a concrete instantiation can be found in appendix C.2 with an IRI reference table in table C.1.

Graph Model

Building on the Semantic Web data model and RDF foundations introduced in section 2.9, we now provide a compact formalization of the graph representation used in this work. Let $\mathcal{I}$ be the set of IRIs and $\mathcal{L}$ the set of literals. Since the subsequent formalization grounds data elements into W3C vocabularies via IRIs, we model graph nodes using IRIs (and omit blank nodes here, as they are not required in the following definitions). An RDF-style triple is

$$(x, p, y) \in \mathcal{I} \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{L}),$$

and a graph is a finite set of such triples

$$G \subseteq \mathcal{I} \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{L}).$$

The graph model above serves as the semantic target representation of the grounding process. In OBDA settings, this graph may be *virtual*, i.e., induced by source data and mappings without full materialization. Unless stated otherwise, we assume no entailment beyond the triples induced directly by the mappings. Where schema-based inference is required, we restrict it to a lightweight RDFS-style fragment (in particular class and property hierarchies, and optionally domain/range typing effects) as formalized in RDF Schema 1.1 [2014].

Vocabulary and grounding into W3C terms

Let $\mathcal{I}$ denote the set of all IRIs. For the subsequent formalization, we distinguish IRIs used to identify entities, properties, and data types, and write

$$\mathcal{I}_E \subseteq \mathcal{I}, \qquad \mathcal{I}_P \subseteq \mathcal{I}, \qquad \mathcal{I}_{DT} \subseteq \mathcal{I}.$$

In addition, let $\mathcal{I}_C \subseteq \mathcal{I}_E$ denote IRIs used as *concepts*. Here, "concept" is used as an umbrella term for the semantic target constraint of an entity-valued label: in the simplest case, it is a class identifier (e.g., `wd:Q11424` for *film*, used as a type), but more generally it may represent a constrained target category used for grounding and entity linking (e.g., `wd:Q2526255`[1] for *film director*, used to restrict candidate entities when linking a value such as `Coppola`). Thus, $\mathcal{I}_{DT}$ contains data type IRIs such as `xsd:gYear`, while $\mathcal{I}_P$ contains property IRIs used for predicates. We assume $\mathcal{I}_{DT}$, $\mathcal{I}_E$, and $\mathcal{I}_P$ to be pairwise disjoint.

Schema-level grounding. We assume a set of schema-level semantic labels of the form $\langle (k, \star), p, t \rangle$ that assign relevant selectors k an RDF predicate $p \in \mathcal{I}_P$ and a target $t \in \mathcal{I}_{DT} \cup \mathcal{I}_C \cup \mathcal{I}_E$. These labels constitute the schema-level grounding of the dataset into the target vocabulary. For example, in our running movie dataset, the selector $k =$ `Director` may be labeled as $\langle$(`Director`, $\star$), `wdt:P57`, `wd:Q2526255`$\rangle$, indicating that values under `Director` participate as objects of `wdt:P57` (director) and are interpreted as entities of type `wd:Q2526255` (*film director*).

[1] Check table C.1 for an IRI reference table.

Instance-level grounding. To relate observed values $v \in \mathcal{V}$ to RDF terms, we assume grounding operators: (i) a (partial) entity-linking function

$$\mu : \mathcal{V} \times \mathcal{I}_C \rightharpoonup \mathcal{I}_E,$$

where $\mu(v, C)$ returns an entity IRI for the entity denoted by v under concept C (if resolvable), and (ii) a typed-literal constructor

$$\text{lit} : \mathcal{V} \times \mathcal{I}_{DT} \to \mathcal{L},$$

mapping (v, d) to a literal with data type d. If $t \in \mathcal{I}_E$, the object is fixed to t (independent of v), which is useful for cryptic codes or boolean flags. If no official IRI exists or linking is undefined, an implementation may mint a local IRI (e.g., `ex:movie/Godfather`) and optionally align it later (e.g., via `owl:sameAs`). For instance-level grounding, we have for example:

$$\begin{aligned}
\mu(\texttt{Godfather}, \texttt{wd:Q11424}) &= \texttt{wd:Q47703} && \text{(film entity)} \\
\mu(\texttt{Coppola}, \texttt{wd:Q2526255}) &= \texttt{wd:Q56094} && \text{(film director entity)} \\
\text{lit}(1973, \texttt{xsd:gYear}) &= \texttt{"1973"\^\^xsd:gYear} && \text{(typed year literal)} \\
\text{lit}(\texttt{2h 55min}, \texttt{xsd:string}) &= \texttt{"2h 55min"\^\^xsd:string} && \text{(raw string literal)}
\end{aligned}$$

For brevity, we constrain directors directly to the occupational concept `wd:Q2526255` (*film director*) and omit the more general typing as `wd:Q5` (*human*), who directed (wdt:P57) a movie.

Definition: A *semantic label* is a triple

$$SL = \langle c, p, t \rangle,$$

where:

- c is a data element, either a schema-level element $(s, k, \star)$ or an instance-level element (s, k, v) (object-abstracted) or (s, r, k, v) (object-specific),
- $p \in \mathcal{I}_P$ is an RDF predicate IRI (e.g., `rdf:type`, or a domain-specific predicate),
- $t \in \mathcal{I}_{DT} \cup \mathcal{I}_C \cup \mathcal{I}_E$ is a target specification: either a data type IRI, a concept IRI, or a fixed entity IRI. Here, a concept IRI (from $\mathcal{I}_C$) serves as a semantic target constraint for entity-valued grounding (e.g., for restricting entity-linking candidates), and is not limited to plain class typing.

Intuitively, the predicate p specifies the semantic role played by the data element, while the target t specifies how values are to be grounded in RDF: as typed literals ($t \in \mathcal{I}_{DT}$), as linked/minted entities under a concept ($t \in \mathcal{I}_C$), or as a fixed entity IRI ($t \in \mathcal{I}_E$). The definition above permits labels both at the schema level ($(k, \star)$) and at the instance level ((k, v) or (r, k, v)). In most mappings, semantics are specified at the schema level and then instantiated per object to produce RDF assertions, which is described in the following.

Deriving RDF assertions

Semantic labels are specified over data elements but are intended to induce RDF assertions. We use parentheses $(\cdot)$ for RDF triples in G and angle brackets $\langle\cdot\rangle$ for semantic labels (mapping annotations about the source). In the remainder of this section, we fix a source $s_0 \in S$ and write $\mathcal{R} := \mathcal{R}_{s_0}$, $\mathcal{K} := \mathcal{K}_{s_0}$, and $D := D_{s_0}$. We also distinguish the roles of selectors: $k_{\text{id}} \in \mathcal{K}$ denotes the selector chosen as object identifier for subject construction, while $k_{\text{attr}} \in \mathcal{K}$ denotes any selector used to populate predicate–object assertions about that subject. For each object $r \in \mathcal{R}$ we construct a subject term $\mathscr{S}(r) \in \mathcal{I}_E$ representing the entity described by r. For instance, for the movie table we may decide $k_{\text{id}} = \texttt{Title}$ so that each row yields a subject $\mathscr{S}(r)$ for the corresponding film, whereas selectors such as $k_{\text{attr}} = \texttt{Director}$ or $k_{\text{attr}} = \texttt{Won_Oscar}$ contribute facts about that film (e.g., `wdt:P57` and `nv:wonOscarYear`).

(a) Subject construction. In practice, semantic labels are typically serialized as mapping metadata (e.g., RML/R2RML in Turtle), while the induced RDF assertions form a separate RDF graph G that may be serialized as well. A semantic label alone is not an RDF triple: it specifies (i) which predicate p to use and (ii) how to interpret values via the target t (datatype, class, or fixed entity). To actually emit a triple, however, one must also provide a subject. The subject-construction function $\mathscr{S}(r)$ supplies this missing RDF resource for each object r; without $\mathscr{S}(r)$, one could state that `Director` maps to `wdt:P57` but could not form a concrete assertion $(_, \texttt{wdt:P57}, \ldots)$ because the subject slot would be undefined. For example:

$$\begin{aligned}\big(\mathscr{S}(r),\ \texttt{rdf:type},\ \texttt{wd:Q11424}\big),\\ \big(\mathscr{S}(r),\ \texttt{wdt:P57},\ \mu(\texttt{Coppola}, \texttt{wd:Q2526255})\big),\\ \big(\mathscr{S}(r),\ \texttt{nv:wonOscarYear},\ \text{lit}(1973, \texttt{xsd:gYear})\big).\end{aligned}$$

Intuitively: *the movie described by r is a film, directed by Coppola, and has `wonOscarYear` 1973.* An *identifier selector* $k_{\text{id}} \in \mathcal{K}$ determines the subject term of an object and its class. This is captured by the schema-level label

$$SL_{\text{id}} \ = \ \langle (k_{\text{id}}, \star),\ \texttt{rdf:type},\ C \rangle, \qquad C \in \mathcal{I}_C.$$

Note that in Wikidata, typing is often expressed via `wdt:P31` (*instance of*) rather than `rdf:type`. For each object $r \in \mathcal{R}$ with $(k_{\text{id}}, v_{\text{id}}) \in D(r)$, let $\mathscr{S}(r)$ be the subject constructed from v_{id} (e.g., via entity linking μ) defined as:

$$\mathscr{S}(r) = \begin{cases} \mu(v_{\text{id}}, C) & \text{if } \mu(v_{\text{id}}, C) \text{ is defined,} \\ m(v_{\text{id}}, C) & \text{otherwise,} \end{cases}$$

where $m(v_{\text{id}}, C)$ mints a local IRI in $\mathcal{I}_E$ for the value v_{id} under concept C. The label induces the type assertion

$$\big(\mathscr{S}(r),\ \texttt{rdf:type},\ C\big) \in G.$$

(b) Predicate population. Given the subject term $\mathscr{S}(r)$ for an object r, attribute labels determine how observed key–value pairs are turned into RDF assertions attached to that subject. Concretely, a schema-level label $\langle(k_{\text{attr}}, \star), p, t\rangle$ (i) fixes the RDF predicate p used in the emitted triple and (ii) specifies how values under k_{attr} are grounded as RDF objects via t. Thus, for each observation $(k_{\text{attr}}, v) \in D(r)$ we emit the triple $(\mathscr{S}(r), p, o)$, where the object $o = o(v,t)$ is constructed as below. An *attribute selector* $k_{\text{attr}} \in \mathcal{K}$ is annotated by a schema-level label

$$SL_{\text{attr}} \;=\; \langle(k_{\text{attr}}, \star),\; p,\; t\rangle, \qquad t \in \mathcal{I}_{DT} \cup \mathcal{I}_C \cup \mathcal{I}_E.$$

For each $(k_{\text{attr}}, v) \in D(r)$, the induced RDF assertion is

$$\big(\mathscr{S}(r),\; p,\; o(v,t)\big) \in G, \qquad o(v,t) = \begin{cases} \text{lit}_t(v) & \text{if } t \in \mathcal{I}_{DT}, \\ \mu(v,t) & \text{if } t \in \mathcal{I}_C \text{ and } \mu(v,t) \text{ is defined,} \\ m(v,t) & \text{if } t \in \mathcal{I}_C \text{ and } \mu(v,t) \text{ is undefined,} \\ t & \text{if } t \in \mathcal{I}_E. \end{cases}$$

For instance, the label $\langle(\texttt{Director}, \star), \texttt{wdt:P57}, \texttt{wd:Q2526255}\rangle$ fixes $p = \texttt{wdt:P57}$, so any director value contributes a triple with predicate `wdt:P57`. We use `nv:` to denote a new vocabulary for predicates not present in Wikidata's vocabulary (see next section). For example, let r be the object for *The Godfather* with $(\texttt{Director}, \texttt{Coppola})$, $(\texttt{Won_Oscar}, 1973)$, $(\texttt{Genre}, \texttt{Crime})$:

1. Datatype target ($t \in \mathcal{I}_{DT}$).

$$\langle(\texttt{Won_Oscar}, \star), \texttt{nv:wonOscarYear}, \texttt{xsd:gYear}\rangle \Rightarrow (\mathscr{S}(r), \texttt{nv:wonOscarYear}, \text{lit}(1973, \texttt{xsd:gYear})) \in G.$$

2. Class target ($t \in \mathcal{I}_C$; entity linking or minting).

$$\langle(\texttt{Director}, \star), \texttt{wdt:P57}, \texttt{wd:Q5}\rangle \Rightarrow (\mathscr{S}(r), \texttt{wdt:P57}, \mu(\texttt{Coppola}, \texttt{wd:Q5})) \in G$$

   (or $(\mathscr{S}(r), \texttt{wdt:P57}, m(\texttt{Coppola}, \texttt{wd:Q5}))$ if μ undefined).

3. Fixed entity target ($t \in \mathcal{I}_E$).

$$\langle(\texttt{Genre}, \star), \texttt{wdt:P136}, \texttt{wd:Q959790}\rangle \Rightarrow (\mathscr{S}(r), \texttt{wdt:P136}, \texttt{wd:Q959790}) \in G.$$

Explicit instance-level labels. Schema-level labels impose a uniform interpretation for all values under a selector. When individual values require disambiguation or correction, e.g.,

- ambiguity (e.g., "Avatar" the word vs the film),
- mixed-type columns/fields,
- entity resolution corrections,
- exceptions or missing values.

One may attach labels to instance-level elements (k, v). This is particularly useful when semantics are encoded as values (e.g., $(\texttt{\$.profession}, \texttt{director}) \in D(r)$ in the JSON example).

A common form is an explicit grounding of a concrete value to a vocabulary concept, e.g.,

$$\langle(\texttt{\$.profession}, \texttt{director}),\ \texttt{skos:exactMatch}^{2},\ \texttt{wd:Q2526255}\rangle.$$

Such labels can refine the default grounding for those values, either overriding it or adding additional assertions, depending on the chosen mapping policy.

5.2.2 Relation to RML (executable semantic labeling).

Our schema-level semantic labels can be seen as an abstract, format-agnostic specification of the same information that is made explicit in an *RML mapping* (and its W3C predecessor R2RML) [RML Community]. RML operationalizes semantic labeling by turning (i) an object selection over a source and (ii) a set of subject/predicate/object construction rules into an RDF graph.

Objects and selectors. RML's `rml:logicalSource` with a `rml:iterator` corresponds to selecting the object set $\mathcal{R}$ (rows, JSON objects, XML nodes, ...). The keys $k \in \mathcal{K}$ in our model correspond to RML references (`rml:reference`) under the chosen reference formulation (e.g., `ql:CSV`, `ql:JSONPath`, `ql:XPath`), i.e., they specify how values are extracted from each object.

Identifier semantics (subject construction). The identifier label $SL_{\text{id}} = \langle(k_{\text{id}}, \star), \texttt{rdf:type}, C\rangle$ corresponds to the `rr:subjectMap` of a `rr:TriplesMap`. Concretely,

- constructing $\mathscr{S}(r)$ from $(k_{\text{id}}, v_{\text{id}}) \in D(r)$ corresponds to an `rr:template` (or `rml:reference`) that builds the subject IRI from the identifier value; and
- asserting the class C corresponds to `rr:class` (yielding `rdf:type` triples; in Wikidata settings, the analogous statement may be expressed via `wdt:P31`).

Thus, the RML subject map is an executable counterpart of our subject construction plus the induced type assertion.

Attribute semantics (predicate population). Each attribute label $SL_{\text{attr}} = \langle(k_{\text{attr}}, \star), p, t\rangle$ corresponds to an RML `rr:predicateObjectMap` with `rr:predicate` p and an `rr:objectMap` (or constant object term) that implements the object construction function $o(v, t)$:

- **Literal-valued** ($t \in \mathcal{I}_{DT}$): `rr:objectMap` uses `rml:reference` k_{attr} and `rr:datatype` t, matching $o(v, t) = \text{lit}_t(v)$.
- **Concept-valued** ($t \in \mathcal{I}_C$): the object term is an entity IRI obtained from the source value under concept t. This can be realized either by (i) constructing a local IRI via a template (our minted $m(v, t)$), or (ii) referencing an IRI produced elsewhere (e.g., via a joined `rr:parentTriplesMap`), corresponding to a defined $\mu(v, t)$.
- **Fixed-entity** ($t \in \mathcal{I}_E$): the object term is the constant IRI t, independent of the source value (e.g., for boolean flags or coded values mapped to a fixed entity).

[2] https://www.w3.org/2004/02/skos/specs, accessed on 02.05.2026

In our formalization, $\mathscr{S}(r)$ is a *function* (defined by the identifier label) that returns an RDF term for object r; it is therefore *not* a node that is stored in G. Rather, for each concrete object r the mapping evaluation produces a subject term $s_r := \mathscr{S}(r) \in \mathcal{I}_E$ (or a blank node), and the emitted triples in G use this term as the subject, e.g.,

$$(s_r,\ \texttt{rdf:type},\ C), \qquad (s_r,\ p,\ o(v,t)).$$

In RML, this corresponds exactly to executing `rr:subjectMap` (via `rr:template`/`rml:reference`) to obtain s_r, while `rr:predicateObjectMap`s then attach the predicate/object assertions to that subject. Listing C.1 shows an RML realization of our labels for the CSV/JSON running example: rows in `movies.csv` are mapped to `ex:movie/{Title}` and typed as `wd:Q11424` via `wdt:P31`, while `wdt:P1476` and `wdt:P57` populate title and director facts. The director link is created by joining `Director` (CSV) with `name` (JSON) using `rr:parentTriplesMap`/`rr:joinCondition`, and Oscar categories are added by iterating over `$.awards[*]` and attaching them to the corresponding movie resource.

5.2.3 Relation to SemTab (CTA/CEA/CPA).

In section 5.7.2 we conduct experiments to empirically assess the ability of LLMs to perform semantic labels on the schema level . The so-called *SemTab challenge* [Jiménez-Ruiz et al., 2020], an annual challenge for tabular data to KG matching collocated with the ISWC conference[3], formulates three core annotation tasks: cell-to-entity annotation (CEA), column-to-class annotation (CTA), and column-pair-to-property annotation (CPA). We view these tasks as a decomposition of automated semantic labeling (see figure 5.1).

CTA as (generalized) key type annotation. CTA generalizes from "column type" to *selector/key type* by assigning a target specification to a selector k via a (partial) typing function

$$\tau : \mathcal{K}_s \rightharpoonup \mathcal{I}_C \cup \mathcal{I}_{DT}, \qquad \tau(k) = t,$$

indicating what values under k represent: either a concept (for entity-valued grounding, $t \in \mathcal{I}_C$, e.g., film/person/place) or a literal data type ($t \in \mathcal{I}_{DT}$, e.g., year/integer/string). Our schema-level labels subsume this information. In

$$\langle (k, \star),\ p,\ t \rangle,$$

the component t plays the role of $\tau(k)$, but the label additionally fixes the predicate p, i.e., *how* values under k are connected in the induced RDF graph.

CPA as predicate selection between selectors. CPA assigns a predicate to an ordered pair of selectors (keys) (k_i, k_j), i.e.,

$$\rho : \mathcal{K}_s \times \mathcal{K}_s \rightharpoonup \mathcal{I}_P, \qquad \rho(k_i, k_j) = p,$$

[3] https://sem-tab-challenge.github.io/2025/, accessed on 03.03.2026

stating that values under k_j should be connected to the subject determined by k_i via predicate p (typically $k_i = k_{\text{id}}$).

CEA as explicit value grounding. CEA provides explicit instance-level grounding of concrete values to KG entities, which corresponds to instance-level labels of the form

$$\langle (r, k, v),\ \texttt{skos:exactMatch},\ e \rangle$$

and can be used to define or correct $\mu(v, t)$ when $t \in \mathcal{I}_C$ (e.g., disambiguation such as "Avatar" the word vs. the film).

5.2.4 Semantic Models

A *semantic model* serves as an interpretation layer between the data layer and the knowledge layer [Pomp et al., 2018]. As illustrated in Figure 5.2, heterogeneous data sources (e.g., CSV tables and hierarchical JSON documents) appear at the data layer and may partially overlap in content. The semantic model itself is represented as a graph whose nodes and edges are (i) *projected* from the knowledge graph and (ii) *extended* with additional classes/properties required by the sources and the application domain. In addition, semantic labels anchor source elements to this graph and thereby determine how RDF assertions are induced from the data. Furthermore, semantic models are dynamic: when users provide sources containing concepts and relationships not covered by the knowledge graph, the semantic model introduces corresponding extensions (e.g., a cinema domain namespace `cd:`) and aligns them to existing knowledge, thereby continuously enriching the knowledge layer [Pomp et al., 2019a]. Hence, the semantic model is not merely a projection but an **interpretation layer**.

Definition: A *semantic model* is a pair

$$SM \ = \ \langle\, G_{SM},\, SL \,\rangle,$$

where G_{SM} is an RDF graph that decomposes into

$$G_{SM} \ = \ G_{\text{proj}} \ \cup \ G_X.$$

Here,

- $G_{\text{proj}} \subseteq KG$ is a projected fragment of the knowledge graph (relevant classes/properties and, if available, lightweight schema constraints), and
- G_X is an extension graph containing triples not previously present in KG. It may contain both schema-level (TBox) extensions (new classes/properties and alignment axioms) and instance-level (ABox) assertions, including assertions induced from the sources by applying SL.

The labels SL anchor semantic knowledge in concrete data elements (bottom layer) and determine how instance-level RDF assertions populate G_X, while G_{proj} provides the projected

conceptual context in which these assertions are interpreted.
In Figure 5.2, G_{proj} corresponds to projected nodes/edges borrowed from Wikidata (prefix `wd`), while G_X captures novel domain additions (prefix `cd` or `nv`) and their alignments.

Example (projected vs. novel triples). For the running example in Figure 5.2, the semantic model contains projected knowledge graph fragments such as

$$(\texttt{wd:Godfather}, \texttt{wdt:P31}, \texttt{wd:film}) \in G_{\text{proj}},$$
$$(\texttt{wdt:P57}, \texttt{rdfs:domain}, \texttt{wd:film}) \in G_{\text{proj}},$$
$$(\texttt{wdt:P57}, \texttt{rdfs:range}, \texttt{wd:film_director}) \in G_{\text{proj}}.$$

In addition, it may introduce application-specific extensions (novel nodes/edges) not previously present in the KG, e.g.,

$$(\texttt{nv:StudioX}, \texttt{wdt:P31}, \texttt{wd:film_studio}) \in G_X,$$
$$(\texttt{nv:ContractX}, \texttt{wdt:P31}, \texttt{wd:Contract}) \in G_X,$$
$$(\texttt{nv:ContractX}, \texttt{nv:with_studio}, \texttt{nv:StudioX}) \in G_X,$$
$$(\texttt{nv:ContractX}, \texttt{nv:billing_type}, \texttt{"per_project"}^{\wedge\wedge}\texttt{xsd:string}) \in G_X.$$

For readability, literal objects induced from source values (e.g., `"per_project"` for `nv:billing_type`) are depicted at the data-source layer in Figure 5.2; formally, they occur as literals in the corresponding RDF assertions in G_X.

5.3 Semantic Type Detection with LLMs

In the following, we consider selected works after the launch of *ChatGPT* in November 2022. While one can make a strong case that (large) language models have existed before this date, we decided to draw a line due to the massive performance increase which was quite suddenly accessible to the public. Most of the works found in this limited range are pre-prints that have not yet been peer-reviewed and published in scientific journals.

Column Type Annotation Korini and Bizer [2023] are among the first to report the application of LLMs for the Column Type Annotation (CTA) task. CTA is a schema-level annotation task that represents a simplified form of our interpretation of semantic labeling (no predicate) as it aims to map the underlying table schema to a conceptualization. In this section, we use the term "semantic type detection" in the narrower sense of column-level semantic labeling. They view CTA as a multi-class classification problem and evaluate different prompt designs. One important artifact that is highlighted is that ChatGPT tends to ignore the instruction to use terms from the label space, and instead answers using different terms. This is a known drawback of contemporary LLMs known as the Hallucination problem [Ji et al., 2023]. Their proposed solutions for this challenge involve first determining a set of classes of the entities

described in the table and depending on this set asking ChatGPT to annotate columns using only the relevant subset of the overall vocabulary. The evaluation of the approach reports competitive performance when evaluated against the more traditional models which are mostly directly fine-tuned for the CTA task and require significant amounts of task-specific training data [Suhara et al., 2022].

Finetuning LLMs Several works [Li et al., 2023b, Hegselmann et al., 2023, Zhang et al., 2023d] aim at customizing LLMs via fine-tuning for tables in particular. The goal is to solve the challenge of table understanding which is closely related to understanding the semantics of a data source as it includes the CTA task for example. Usmani et al. [2023] highlight the importance of multi-modal knowledge graphs, present a review of the current state, and propose an ontology towards further development. Furthermore, since important use cases for datasets can be attributed to numerical data, it is important to have solid numerical reasoning skills [Gottschalk and Demidova, 2022]. Several works suggest that modern LLMs excel in simple problem settings, but they fall short of human expert performance in problems requiring numerical reasoning over long contexts. As the complexity of challenging mathematical problems increases, LLMs currently exhibit suboptimal performance [Chang et al., 2024, Zhao et al., 2023].

Knowledge Graph engineering with LLMs is investigated in several works [Allen et al., 2023, Meyer et al., 2023a, Frey et al., 2023]. Here the goal is to utilize the LLM for common tasks related to KGs. For example, Meyer et al. [2023b] investigate SPARQL query generation as well as knowledge extraction from fact sheets and KG exploration among others. They present several prompts that essentially test how to pose questions in natural language executed against serialized KGs. The experiments show that LLMs can return syntactically correct SPARQL queries and even entire serialized RDF models with the desired form based on a task formulated in natural language. Further, LLMs can find relationships in KG and answer basic questions correctly, e.g., *"Are there any connections between US and UK?"*. They report major performance differences between GPT-3.5 and GPT-4 in favor of the latter. One particular prompt aims at extracting knowledge from tables and converting it into a serialized KG, which is very close to the idea of semantic modeling. The experiment illustrates several problems with the output of contemporary LLMs:

- A tendency to prioritize the usage of *schema.org* vocabulary. While this works well for well-known entities and properties, the LLMs invent reasonable, but non-existent classes and properties (in the *schema.org* namespace) for concepts and relations that are too specific.
- Non-deterministic output: For multiple runs of the same prompt to the LLM, the output varies.
- Invention of non-existent properties, prefixes, and classes: If the LLM cannot identify a fully matching class for a concept or a relation, URIs for those elements are invented for the raw RDF output. While this would be possible in its own namespace, the classes and properties are placed in existing namespaces resulting in the URIs not being resolvable.
- Non-functional queries: SPARQL queries generated by GPT-3 did not return the expected

LLM		KG	
Pros +	**Cons -**	**Pros +**	**Cons -**
General knowledge	Implicit knowledge	Structural knowledge	Incompleteness
Language processing	Hallucinations	Accuracy	Lacking language understanding
Generalizability	Indecisiveness	Decisiveness	
	Black-Box	Interpretability	Unseen facts
	Lacking domain knowledge	Domain-specific knowledge	
		Evolving knowledge	

Table 5.2: The pros and cons of LLMs vs KGs as described by [Pan et al., 2024]

results when executed against a knowledge graph, albeit being syntactically correct.

To conclude, the results obtained by Meyer et al. [2023b] show that the problems commonly observed with LLMs also limit their ability to conduct tasks in the semantic domain. It is therefore not possible to use LLMs out-of-the-box for semantic relation inference. Since semantic type detection is simpler than semantic relation inference and also relies heavily on obtaining context, and hence this area of automation is investigated more closely.

5.4 Using Knowledge Graphs with LLMs for Semantic Model Creation

This section addresses RQ3 by examining how LLMs can support the automated construction of semantic models in SEDAR, which in turn form the basis for generating mappings and enabling ontology-based access. Considering the findings from the previous section, it becomes clear that LLM outputs cannot be used directly in SEDAR and must be manually verified, constrained, or transformed before they can support semantic model creation (semantic refinement). In the following, approaches on how to utilize LLM-generated output in automated semantic model creation are conceptualized.

Complementary Strengths of LLMs and Knowledge Graphs To understand how LLMs can assist semantic model creation, it is necessary to examine their strengths and weaknesses in relation to knowledge graphs. Although the technologies for linked data have become more mature in recent years, the amount of data considered in Semantic Web applications is far less than in Big Data applications [Haller et al., 2022]. This indicates that scalability to large, heterogeneous data sets is a major challenge. Here, LLMs can potentially be a great help.

However, even though LLMs can effectively possess rich knowledge learned from massive amounts of training data and benefit downstream tasks at the fine-tuning stage, as previously described, they still have significant limitations due to the lack of external knowledge [Chang et al., 2024]. In contrast, KGs are structured knowledge models that explicitly store rich factual knowledge, but they are difficult to construct and evolve by nature, making it challenging to generate new facts and represent unseen knowledge [Iglesias et al., 2023].

Therefore, it is reasonable to view KGs and LLMs as two complementary technologies

whose integration has the potential to produce synergy, capitalizing on the strengths of each while mitigating their respective weaknesses.

Limitations of LLMs in Handling Graph-Structured Knowledge These observations motivate a closer look at established research on combining the technologies, particularly with regard to complementary reasoning capabilities. Pan et al. [2024] present a detailed discussion on research towards the unification of language models and KGs, where they contrast the pros and cons of (large) language models vs KGs. Table 5.2 confirms the previous findings about the drawbacks of LLMs, namely hallucination, a lack of domain-specific knowledge, indecisiveness, and a lack of interpretability. As the most basic semantic unit, entities play a crucial role, and incorporating their knowledge into LLMs helps to improve semantic understanding. In addition, there are also a large number of relational triples in the KG, which can provide sufficient structured information to improve the semantic understanding further. Since conventional LLMs trained on plain text data are not designed to understand (graph-)structured data such as knowledge graphs, they might not fully grasp or understand the information conveyed by the KG structure.

Conversely, the automated construction of KGs is equally challenging, and current approaches to KGs are inadequate in handling the incomplete and dynamically changing nature of real-world KGs. Additionally, many of the current techniques for KGs are tailored to particular tasks and, therefore not easy to generalize to broader applications. This suggests that KGs and LLMs indeed complement and may synergize Pan et al. [2024].

Approaches to Integrating LLMs and Knowledge Graphs

So-called *KG-enhanced LLMs* [Hu et al., 2023] integrate KGs into LLMs to address the LLM's difficulty in recalling factual knowledge while generating knowledge-grounded contents. Yang et al. [2024a] review recent developments in this area and categorize them into before-, during-, and post-training enhancement according to the stage at which KGs participate in training.

An example of **training enhancement before and during training** is *K-BERT* [Liu et al., 2020b]. Before training the input text is expanded into graph structure to merge the structured knowledge of KGs and then converts the merged graph back into text for training. It converts input texts to knowledge-rich sentence trees by injecting related triples and fusing them with relevant subgraphs of the KG. For example, an original input sentence such as *"Tim Cook is currently visiting Beijing now"* is expanded by further sentences such as *"Tim Cook is the CEO of Apple"*, *"Bejing is the capital of China"* and "*Bejing is a City*" through factual knowledge coming from a KG. The resulting sentence tree reads, *"Tim Cook CEO Apple is visiting Beijing capital China is a City now"*.

An example of post-training enhancement is *KG-Rank* [Yang et al., 2024b], where the authors leverage a medical KG with ranking techniques, aiming to improve free-text question-answering in the medical domain. While the example comes from the medical domain, it illustrates a general pattern that is also relevant for link text: LLMs tend to hallucinate missing do-

main knowledge, whereas KG-enhanced models remain aligned with factual constraints. Upon receiving a question, they initially retrieve factual information from a medical KG and then apply a ranking to obtain the most relevant triplets which then, combined with the task prompt, are input into LLMs for answer generation. To illustrate the potential benefits the authors consider an example to address hallucinations by *LLaMa2-13b*:

Question

"What would be the ideal diet recommendations for a 53-year-old male with acute renal failure and hepatic failure?"

LLaMa2-13b answer

"... Protein intake: Provide adequate protein to support liver function and wound healing. The recommended daily intake of protein for a 53-year-old male is approximately 1.6-2.2 grams per kilogram of body weight..."

KG-enhanced LLaMa2-13b answer

"Low protein intake: Protein intake should be limited to about 0.8-1.0 gram per kilogram of body weight per day, as excessive protein intake can exacerbate renal failure and liver disease..."

These examples highlight the potential of combining LLMs' language understanding capabilities with the rich, structured knowledge in KGs to produce more reliable and contextually relevant responses, thereby addressing the limitations of each technology when used independently. In the context of this dissertation, these insights motivate the subsequent integration strategies in SEDAR, where in section 6.2 we present a post-training enhancement for data science agents based on KGs.

5.5 Semantic Modeling Platforms

This section addresses RQ2 and provides the background necessary for RQ3. Since SEDAR builds on semantic models as the central abstraction for enabling ontology-based access and integration, it is essential to understand how existing platforms support the creation, refinement, and evolution of such models. The following survey establishes the technological context for design decisions, showing where current platforms succeed, where they fall short, and which gaps motivate the contributions developed later in this dissertation.

For the purposes of this dissertation, **semantic modeling platforms** define the state of the art against which SEDAR's modeling workflow must be positioned and extended. A semantic modeling platform is a system that offers a graphical user interface for inspecting, validating, and refining (refinement phase) the outcomes of automated semantic labeling and relation inference methods. It also supports manual creation of semantic models [Paulus, 2024]. Hence,

we address the second, third and especially the fourth step of figure 5.1.

Synthesis of Existing Approaches *ESKAPE* [Pomp et al., 2017] is a semantic data platform handling the full data management process from ingestion to extraction supporting heterogeneous data sources. It introduces an intuitive interface where users can create semantic models. In [Pomp et al., 2018], the authors report the integration of ESKAPE into a data lake in a production setting. Instead of just dumping raw data into the data lake directly, they use ESKAPE during data ingestion to enable the semantic annotation of data sets to perform semantic data integration. To combine the semantic models, ESKAPE allows the integration of external KGs, such as WordNet, BabelNet, or domain-specific ontologies. Elements of the KGs can be reused during the modeling process and therefore serve as anchors when relating different distinct semantic models and data sets.

KARMA [Gupta et al., 2012] is a data integration tool that enables users to ingest data as well as a vocabulary defined as an OWL ontology to create a corresponding semantic model. KARMA supports multiple data formats and proposes an initial semantic model which the users can edit using a GUI. KARMA is an important pioneering work being one of the first to introduce key technologies such as conditional random fields and an algorithm to solve a Steiner tree problem for learning of semantic labels and to discover relationships among the schema elements of a source. Once the semantic modeling based on the ontology is completed, KARMA integrates data sets based on the defined semantic model. The resulting semantic models can be exported as a RDF file. In contrast to ESKAPE, KARMA provides initial semantic labeling suggestions, but it also requires users to have in-depth knowledge about technical details, e.g., the RDF format. Furthermore, it focuses on automatically creating semantic models based on a fixed ontology per use case where the ontology does not evolve over time. ESKAPE claims to be easier to use because it hides the technical details; however, there is no implementation available to compare the usability.

Recently, the open-source platform *PLASMA* [Paulus et al., 2021b] has emerged from ESKAPE. Paulus et al. introduce a phase called *semantic refinement* in which the human is responsible for improving the quality of the semantic model by validating, correcting, selecting, or exchanging concepts. The semantic models can be created, deleted, searched, and exported as RDF in Turtle syntax. A central idea of the framework are continuously evolving ontologies. As sources need to be added or updated, the created semantic model must be extended or adapted to reflect these changes. To address this issue, Pomp et al. [2019b] propose an approach featuring an evolutionary KG that consists of an internally growing universal KG and data source-specific mappings. A user, who creates a semantic label, is enabled to define and use concepts and relationships in their semantic models previously unknown to the KG. This is, for instance, the case if the concept the user wants to use is missing in the current conceptualizations. This novel knowledge must be integrated into the ontology of the universal KG. For this, Pomp et al. developed a strategy for the controlled and evolutionary development of the KG which evaluates the relationships and concepts that a user selects or introduces, and validates if the user introduces contradictions to the current KG. PLASMA displays a mature

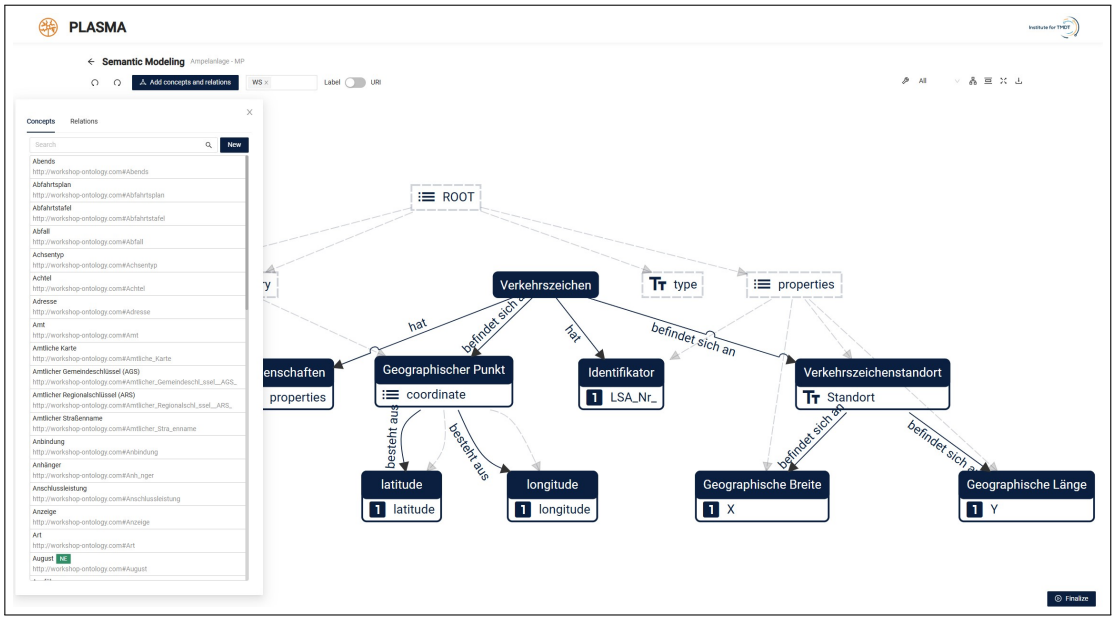


Figure 5.3: Screenshot of the modeling interface in PLASMA [University of Wuppertal, 2025]

solution, proven to be applicable in production with many built-in features.

Figure 5.3 presents a screenshot of the modeling view. PLASMA allows users to import arbitrary ontologies. New semantic elements can be added to the model via the left-side drawer, which lists all available ontology elements. These can be dragged onto the canvas to create new entities within the semantic model. PLASMA's export feature ensures interoperability by allowing users to generate RDF models compatible with external knowledge graphs and data integration platforms.

The dataspace management system On-demand Data INtegration (*ODIN*) [Nadal et al., 2019] allows data integration by virtually querying heterogeneous data sources grounded on KGs. ODIN automatically extracts the schemata from (semi-)structured data sources, translates them into a canonical data model, aligns their schemata, and generates target-specific metadata from them. In the spirit of semantic refinement, it relies on a supported user feedback process for incrementally merging the semantics into a dataspace-wide ontology. ODIN supports an advanced visual query mechanism by relying on *WebVOWL*, which is an open-source application for interactive visualization of ontologies. This ontology-mediated query interface allows selecting nodes of interest from any available semantic model by marking them graphically to generate a SPARQL query. Data sources associated with the marked nodes represent the connection point to the data sets; wrappers encode the query to extract their data and expose a first-normal form relation of their schemata.

MantisTable, introduced by Avogadro and Cremaschi [2021], is a semantic modeling platform designed to annotate tabular data, making table annotations more accessible and interpretable for users. The annotation process is structured into five key phases: During the data preparation, input data can be preprocessed by removing unwanted elements such as HTML tags and stop words. The column analysis phase applies regular expressions to classify lit-

eral columns by scanning their values, while non-literal columns are identified as named entity columns. The S-column, representing the dataset's main subject, is determined based on statistical features. In the annotation phase, semantic labeling is performed using DBpedia as the ontology, with an internal semantic type detection algorithm (ABSTAT [Spahiu et al., 2016]) to assign data types to literal columns. The predicate annotation phase focuses on inferring semantic relationships between columns. Finally, in the entity linking phase, cell contents are mapped to corresponding entities within a knowledge graph. Each phase begins with an automated step, presenting preliminary results to the user for verification and correction. However, MantisTable is specifically tailored for tabular data and does not support custom ontologies. Over time, the platform has evolved through five versions. The latest iteration introduces a redesigned user interface that, like KARMA and SAND, enhances visualization by overlaying the semantic model as a graph on top of the tabular representation.

SAND, a semantic modeling tools, was introduced by Vu and Knoblock [2022]. Like MantisTable, it focuses exclusively on annotating tabular data but introduces a project-based management system that enables users to organize multiple data sources efficiently. Upon uploading a dataset, SAND presents both the table's structure and an interactive graph-based visualization of the semantic model, similar to KARMA and MantisTable. A key feature of SAND is its ability to integrate automated modeling steps, which assist in generating an initial semantic model. Additionally, it supports entity linking to external knowledge graphs. Unlike other tools that apply automation by default, SAND offers it as an on-demand feature, giving users full control over when and how automated processes are executed. Once an automated suggestion is generated, users can review, refine, and correct the model by adding missing relations between previously unmapped columns or adjusting incorrectly linked entities. Beyond annotation, SAND provides direct RDF transformation capabilities and allows users to export mapping configurations in D-REPR [Vu et al., 2019b] format, making it adaptable for integration with external mapping tools, particularly for handling large-scale data sources.

Comparison Criteria To systematically assess how well existing systems support the contemporary requirements, we define the following comparison criteria. These criteria reflect the capabilities most relevant for SEDAR such as automation, ontology evolution, interoperability, and mapping readiness. In table 5.3 we review complete modeling platforms, that include methods for semantic labeling and/or modeling, as well as a graphical user interface to allow the user the adaptation of the created artifacts. For this, we first introduce criteria to compare and to check the comprehensiveness of the existing proposals in the literature:

1. **HR**: *Hierarchical data*, does the platform support only tabular or also hierarchical data?
2. **OE** *On-demand ontology extension*, creating new elements during the semantic model creation in case they are not available in an ontology
3. **SR:** *Semantic relationships*, any form of hierarchical, generic, or pre-defined semantic relationships (semantic connections between data sets, e.g., for provenance or governance). It should be noted that none of the existing data lake systems provides advanced modeling features such as those depicted in Figure 5.2.

Table 5.3: Comparison of semantic modeling platforms.

	HR	OE	IAM	SR	EX	M	C	O
ESKAPE [Pomp et al., 2018]	✓	✓	✓	X	✓	✓	X	X
KARMA [Szekely et al., 2015]	✓	X	X	✓	✓	✓	✓	✓
PLASMA [Pomp et al., 2021]	✓	✓	✓	✓	✓	✓	✓	✓
ODIN [Nadal et al., 2019]	✓	X	✓	✓	✓	X	✓	✓
SAND [Vu and Knoblock, 2022]	X	X	✓	✓	X	✓	✓	✓
MantisTable [Avogadro and Cremaschi, 2021]	X	X	✓	✓	X	X	✓	✓

4. **MI:** *Metadata interoperability* This denotes whether the authors have considered the possibility of importing or exporting metadata. i.e., serialization of the semantic model. This is strongly related to criterion **C** since serialized models are ported to different systems more easily.
5. **IAL:** *Initial automatic semantic labeling* denotes the functionality to generate semantic labels automatically from the schema at the time of ingestion.
6. **C:** *Compatibility with Semantic Web technology*. This criterion checks whether semantic labels are modeled in a language of the Semantic Web (e.g., RDF, OWL).
7. **O:** *Open Source*. This criterion checks whether there is an open-source implementation available.
8. **IAM:** *Initial automatic semantic model*. In contrast to IAL, this denotes here the functionality to generate entire semantic models automatically from the schema after the ingestion of a data source. This includes the first three phases in Figure 5.1.
9. **EX:** *Support for external KGs*. Is the approach capable of handling arbitrary KGs or fixed to a predefined KG?
10. **M:** *Maturity*. Here we check whether the system has been reported to be successfully utilized in a Big Data project.
11. **C:** *Compatibility with Semantic Web technology*. This criterion checks whether semantic labels are modeled in a language of the Semantic Web (i.e., RDF or OWL).

Taken together, these systems illustrate the evolution from purely manual modeling interfaces to partially automated pipelines with varying degrees of semantic recommendation, ontology evolution, and schema integration. However, none of these platforms provides a unified approach that combines semantic modeling, heterogeneous data understanding, and automated mapping generation and especially semantic data access as envisioned for SEDAR.

5.6 Ontology-based Data Access & Integration

Having introduced semantic labels and semantic models as conceptual representations of heterogeneous data sources, we now examine how these semantics can be operationalized for unified querying and integration. Mami et al. [2019b] discuss candidates for a 'universal' query language and based on their findings exploring various query translation methods, they see SQL and SPARQL as the most suitable languages to act as a 'universal' language for realizing heterogeneous data integration. A core distinction in this research concerns how the semantics are leveraged for access: by translating queries on the fly, or by materializing data in a semantic format. This subsection outlines these two paradigms.

5.6.1 Query Rewriting vs. Materialization

A challenge in this scenario is the combination of different abstraction levels and formalisms. Whereas ontologies model the conceptual level, use open-world assumption, and are based on description logics, the data sources are described by schemas on the logical level which focus on the structure of the data and use close-world assumption. Here, Fathy et al. [2019] identify two main approaches:

1. **rewriting (or on-demand translation) of queries** (usually expressed in SPARQL) into the required query language of the data sources (see Figure 5.4) or
2. **materialization**, meaning all source data sets are converted into a common data format (e.g., RDF or relational tables) and stored in one data repository.

Thus, the main difference is whether to translate the data or the query. This can be interpreted as the tradeoff between the ETL/ETL principles. Because SEDAR follows the traditional data lake design principles, **this dissertation focused on on-demand translation**.

Query rewriting is part of the larger domain of data federation [Gu et al., 2022] which addresses the problem of uniformly accessing multiple, possibly heterogeneous data sources, by mapping them into a unified schema. This is the topic of **ontology-based data access** (OBDA) which has been developed since the 2000s [Poggi et al., 2008] to facilitate access to various types of data sources. OBDA uses an ontology as the common data model and a set of existing data sources form the data layer. The goal is to build a service on top of this, aiming at presenting a conceptual view of data to the clients. By posing the query to the ontology, the primary advantage is that users can query various data sources without knowing how the data is organized and where it is stored. In OBDA specifically, the conceptual view is expressed in terms of an ontology, that will represent the unique access point for the interaction between the clients and the system for a collection of data sources that exist autonomously and have not been necessarily structured to store the ontology instances. The ontology describes the domain of interest and is usually developed independently of the data layer. The common data model used for querying can be also expressed in other schema languages. Earlier approaches for data integration used the relational model and translated the data sources into a common

relational schema [Lenzerini, 2002].

Materialization Calvanese et al. [2018] introduced the term **ontology-based data integration** (OBDI) to emphasize that data from multiple sources is accessed *and* integrated. In OBDA, the data remains in its original formats, and queries are translated to align with the source formats, enabling virtual access without physically moving the data. In OBDI, which also leverages conceptual modeling to provide data, but first requires the data to be integrated into a new storage system. This follows the ETL paradigm, where data is converted into a unified format before retrieval. The transformed data is then stored in a centralized location, from which it can be accessed by consumers. Since the data has already been transformed, no additional processing is required during retrieval. The first stage in Figure 5.5 illustrates this process. Like OBDA, it begins with creating mappings from the original data sources to an ontology. However, in OBDI, the data is transformed according to the formalized mappings and stored in a data repository optimized for the target format. Consumers can then query this storage using ontology-based query languages, such as SPARQL. The data is returned directly in the required format, as it has already been preprocessed to align with the query structure. In the first stage of Figure 5.5, the illustrated data transformation results in a materialized RDF graph, that can be uniformly queried using SPARQL. *SDM-RDFizer* and *RocketRML* [Iglesias et al., 2020, Simsek et al., 2019] are concrete solutions that built upon RML and follow the materialized approach.

Tradeoff Both OBDA and OBDI offer distinct benefits and limitations. OBDA allows for flexibility, as it performs data transformation dynamically with each query, making it better suited for environments where data frequently changes. However, OBDI requires immediate integration of data into the new storage system after each update, which can become resource-intensive if updates occur frequently. Additionally, in OBDI, data is stored redundantly, introducing the challenge of keeping everything synchronized. If the ontology or data formats change, a full update of the storage system is required, increasing the workload.

The advantage of OBDI is that this effort is handled in advance, reducing query response times since no additional transformation is necessary during retrieval. This is particularly useful when consumers frequently make similar requests, as it eliminates redundant transformation processes and reduces system load. While caching mechanisms can alleviate some load in OBDA, their efficiency depends on the complexity of the queries, the structure of the ontology, and the quality of the mapping.

In both cases, the creation of high-quality mappings in particular can be tedious and time-consuming [Hoseini et al., 2023], which is why this dissertation puts special emphasis (sections 5.3 to 5.5 and 5.7) on these phases (figure 5.1).

5.6.2 The OBDA Architecture: Ontology, Mappings, and Query Translation

Having outlined the two main access paradigms, we now turn to the standard OBDA architecture, which clarifies the roles of ontologies, mappings, and query translation in virtual knowledge graph systems.

Query rewriting in this relational setting has been well studied; we consider the more complex scenario of query translation with ontologies over heterogeneous sources. Query translation [Mami et al., 2019b] is hence strongly related and essential for any effective OBDA framework to deal with the variety of heterogeneous data sources in a KG. In OBDA, on-demand query translation exposes the content of data sources as RDF triples, using classes and predicates from a KG. The RDF triples are not materialized, they are part of a virtual knowledge graph (VKG), which means that data remains in the data sources instead of being stored in some common database. A VKG system has the following components (see Figure 5.4):

a) a collection of **data sources**,
b) **queries** that describe user information needs,
c) an **ontology** (or in practice a KG) and
d) **mappings** that encode the connection from the ontology to the data sources.

The W3C published recommendations for languages for components (b)-(d): SPARQL, OWL 2, and R2RML[4], respectively and SQL is the language for relational DBMSs. Despite the apparent similarities between SPARQL and SQL, the two languages have significant differences that need to be respected by any OBDA system to transform SPARQL queries into SQL. Map-

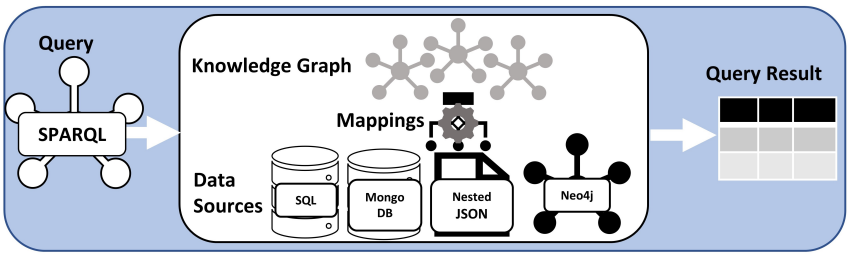


Figure 5.4: Illustration of Ontology-Based Data Access [Hoseini et al., 2024d]

ping creation and management is probably the most complicated OBDA design-time task, as the mapping specifies the semantics of the data sources in terms of the ontology and bridges the typically large conceptual gap between the source schema and the ontology. By applying knowledge representation and automated reasoning techniques, an OBDA system uses ontologies and mappings to reformulate the SPARQL queries into source-specific queries.

While there exists a bulk of literature on OBDA over relational data, the situation looks different for non-relational data models. There are four main groups of *NoSQL* (Not Only SQL) DBMSs [Fathy et al., 2019]: document-oriented, column-oriented, graph-oriented, and key-value stores. Each of these DBMSs relies on different data models and query engines, which complicates the task of creating a uniform OBDA system in this case. Because of increasing popularity and the lack of unified query languages and data models for these systems, there is

[4] https://www.w3.org/TR/r2rml/, accessed on 02.05.2026

growing interest in applying OBDA in this context. Systems based on VKGs promise to provide the desired unified semantic view of data sets. Early papers include [Curé et al., 2011, 2013] and different languages have been proposed to extend the R2RML mapping language.

RML, Function Extensions, and Human-Readable Mapping Languages Since mappings are the core design-time artifact in OBDA, considerable research has focused on mapping languages and tools. This subsection reviews the main developments in this area. The evolution of mapping languages demonstrates how the community addressed heterogeneous sources and usability concerns.

The RDF Mapping Language[5] (RML) [Dimou et al., 2014] is such a proposed extension to deal with heterogeneous data sources. RML is a mapping language that broadens the scope of R2RML by enabling it to handle data from various structured formats, such as CSV, XML, and JSON, in addition to relational databases. RML follows exactly the same syntax as R2RML; therefore, RML mappings are themselves RDF graphs. While R2RML is a W3C standard, RML itself remains a community-driven specification without official W3C endorsement.

There are proposals to extend RML with functions and operations described uniformly, unambiguously, and independently of the technology used for implementation. Examples of those include the Function Ontology (FnO) [Meester et al., 2017] and FunUL [Junior et al., 2016]. While RML extends R2RML to schema transformations, the combination of RML with FnO extends R2RML concerning data transformations. RML is widely spread and implemented in existing systems, and many proposals exist that aim to ease its use. YARRRML [Heyvaert et al., 2018b] simplifies the use of both R2RML and RML by a human-readable text-based representation for mapping rules expressed in YAML[6], a widely used human-friendly data serialization language. The RML Editor [Heyvaert et al., 2016], RMLx [Aryan et al., 2017] Map-On (only for R2RML [Sicilia et al., 2017]) and MapVOWL [Heyvaert et al., 2018a] provide visual support to help users in generating Linked Data from raw data by defining RML mappings. In [García-González et al., 2020] the authors conduct a usability experiment on three different languages: ShExML [García-González et al., 2020], YARRRML and SPARQL-Generate [Lefrançois et al., 2016]. RML was left out, because of too high verbosity. Their prototype translates files (JSON, XML) into RDF using one of the mappings languages and examines the usability via a questionnaire for first-time users, which have only basic skills in programming and linked data. The results show that ShExML users tend to perform better than those of YARRRML and SPARQL-Generate.

xR2RML is a language for mapping various types of databases to RDF which enables the translation of a broad scope of data sources via query re-writing [Michel et al., 2019]. To achieve this goal, they propose a two-phase approach. First, they define an abstract query language derived from SPARQL. Utilizing the xR2RML mapping language and leveraging R2RML-based SPARQL-to-SQL works, they introduce a generic method to translate a SPARQL graph pattern into their abstract query language. In the second phase, the abstract query is translated into the

[5] https://rml.io/specs/rml/, accessed on 02.05.2026
[6] https://yaml.org/spec/, accessed on 02.05.2026

query language of a target database. For demonstration purposes, they apply the approach to MongoDB but acknowledge that adopting a different NoSQL DBMS will still be challenging. This is mainly because these systems are generally optimized for fast storage and retrieval of vast collections of data, rather than on expressive query languages. The prototype implementation is open-source[7] including the SPARQL-to-MongoDB translator and connectors for the MySQL and PostgreSQL relational databases.

R2RML has been proven to be a useful mapping language for relational sources. Simply extending the R2RML standard to support other types of sources does not necessarily carry on all its features [Chortaras and Stamou, 2018]. For example, select conditions and transformation functions are supported by R2RML implicitly relying on the expressivity of SQL, but this cannot be applied for querying XML or JSON documents. In [Corcho et al., 2020], the authors argue first, typical mapping languages are designed to work with a specific data format, and second, they are designed in a format to be parsed by machines and they do not take into account human readability and compactness. YARRRML is a contribution in that direction. Because these languages are not necessarily interoperable, and many of them support a specific engine only, the next generation of OBDA systems would have to consider how to address translation mapping languages, in addition to the query re-writing techniques that have been widely addressed in the state of the art of OBDA so far (see Figure 5.5).

5.6.3 Historical Evolution of OBDA and Summary

To contextualize current research challenges, this subsection summarizes the historical development of OBDA and traces how the field evolved from early materialization-based systems to modern virtual knowledge graph platforms. We identify four different periods based on the existing literature (see Figure 5.5):

First, there are proposals following the materialized approach, i.e., transforming all data sources into a huge RDF graph to be queried using SPARQL. Although the approach might suggest a uniform query method for heterogeneous data sources, the question of data integration remains unanswered as data is only transformed into a common format, but it is unclear how they relate semantically to each other. The approach also has the previously described deficiencies about the liveliness of data and the overhead that comes with the initial transformation into RDF.

These weak points were then addressed by the first platforms to perform OBDA over relational databases via query re-writing, which leaves the data in their respective DBMS retrieving it through the native via query optimization techniques. Although these solutions display impressive progress, they are mostly limited to supporting only a single relational DBMS at once (indicated by a red bar). The process of creating mappings is also mostly manual and hence tedious, error-prone, and tailored towards a single static ontology.

The third period bridges the gap to access NoSQL databases and more flexible data models such as nested structures, graphs, or REST APIs and also expands the query result space

[7] https://github.com/frmichel/morph-xr2rml/, accessed on 02.05.2026

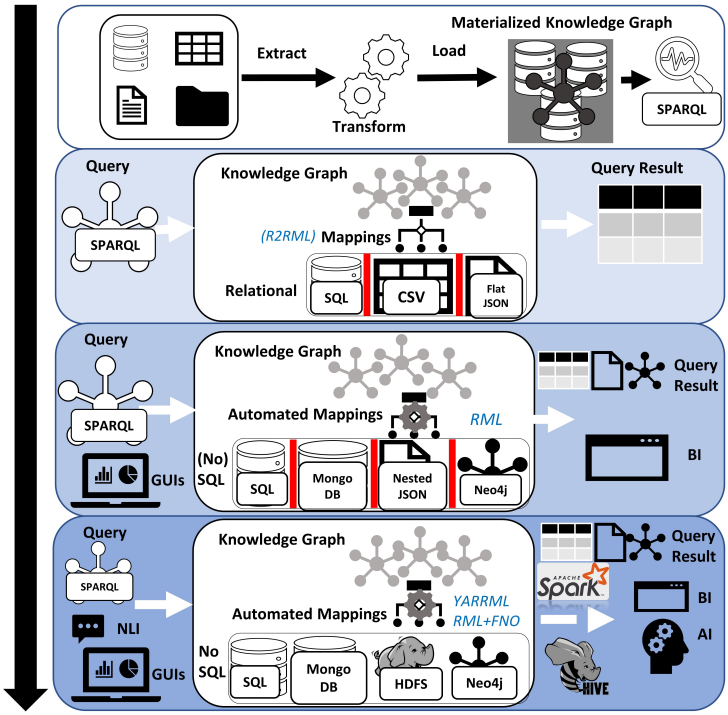


Figure 5.5: Timeline of OBDA techniques [Hoseini et al., 2024d]

to NoSQL data models. In this period many 'single-purpose' solutions arise that extend support to a single NoSQL DBMS or semi-structured files like JSON and XML, but no solution exists that is flexible enough to query arbitrary databases. In this period we also observe the first real-world applications that built GUIs for these methods to perform business intelligence effectively, e.g., [Kharlamov et al., 2017]. In parallel, the development of algorithms for automated generation of mappings gained increasing momentum.

Finally, the most recent approaches add efficient, scalable, and distributed processing engines. Furthermore, we observe the emergence of data-driven methods, in particular scaling usability of ML analytics, which requires managing a large variety of heterogeneous data with a unified mechanism, to save time on developing new ML solutions, and to reuse already developed ones. For this, companies commit to the development of long-term common semantic frameworks for industrial data management and ML analysis [Zhou et al., 2021].

In summary, OBDA provides a principled framework for querying heterogeneous data using a shared conceptual model. However, its practical deployment critically depends on high-quality mappings, which remain costly and labor-intensive to construct. This gap directly motivates the next section and our answer to RQ3, where we examine how semantic models and LLM-based automation in SEDAR can support or partially automate the mapping process, lowering the barrier for applying OBDA in industrial settings.

5.7 Implementation in SEDAR

To equip SEDAR with semantic data management capabilities required for answering RQ3, we integrate three architectural components: ontology/KG management, semantic modeling (split into LLM-based semantic labeling and GUI-based model creation), and ontology-based data access.

5.7.1 Ontology & KG Management

The ontology module provides the conceptual backbone for SEDAR. Its role is not merely to import OWL files, but to maintain a consistent semantic view across heterogeneous datasets and evolving workspaces. This capability is essential because semantic labels, mappings, and OBDA queries depend on a stable yet extensible conceptual layer that reflects the domain knowledge available in the system.

To support semantic annotation of *Datasets*, each *Workspace* is initialized with a default ontology. Users may replace or extend this ontology to reflect domain-specific conceptualizations, enabling SEDAR to adapt to different organizational or industrial contexts. Conceptually, SEDAR enforces that at least one ontology remains active in every workspace to ensure the operability of ontology-dependent components such as semantic labeling, modeling, and querying.

Ontologies added to a workspace are stored and managed through *Apache Jena Fuseki*, which functions as internal triple store. Instead of focusing on file formats or storage details, the architectural principle is that all ontologies regardless of origin are integrated into a uniform semantic layer that exposes their vocabulary, classes, and properties to downstream components.

A further requirement was to provide users with an intuitive understanding of the conceptual structures that underpin their semantic models. For this purpose, SEDAR integrates `WebVOWL` [Lohmann et al., 2014] as an ontology visualization component. The system presents users with an interactive graph representation of the ontology currently active in the workspace, supporting tasks such as exploring class hierarchies, inspecting relationships, and verifying the semantic context used during modeling. All communication with the triple store is routed through the backend, ensuring controlled access and consistent system behavior.

Figure B.7b illustrates the resulting visualization, which allows users to navigate the ontology and inspect vocabulary elements relevant for semantic labeling and modeling.

5.7.2 Creating Semantic Labels with LLMs in SEDAR

To support early phases of semantic modeling, SEDAR provides an optional LLM-based semantic labeling. This component proposes initial ontology-based labels for dataset attributes and is designed to reduce user workload, especially when dealing with large or unfamiliar datasets. The labeling procedure relies on three conceptual steps:

1. extracting representative samples of dataset content,

Algorithm 1: Semantic Labeling Workflow

Input: plasma_id, ontology
Output: semantic labels
Retrieve all modelings and find matching modeling for `plasma_id`
Retrieve all ontologies for `workspace_id` and find matching ontology
Get updated model for `plasma_id` and extract dataset IDs
foreach *dataset_id* **do**
    Retrieve dataset information;
    Prepare a sample of the dataset;
    Call `predict_semantic_labels` to get labels;
    Inside `predict_semantic_labels`:
        Prepare dataset and ontology concepts for prediction;
        Send concepts to a prediction service;
        Receive predicted labels;
    Process the predicted labels;
    Call `get_annotation` to retrieve annotations;
    Inside `get_annotation`:
        Query the ontology to retrieve annotation for the label;
        Return annotation if found, else return no annotation;
    Call `create_semantic_label` to create labels;

2. generating structured prompts that combine dataset samples with ontology concepts, and
3. validating LLM predictions against the ontology through SPARQL queries.

This ensures that the generated labels remain grounded in the active ontology and mitigates the risk of hallucinations. The workflow is summarized in Algorithm 1, which shows how predictions are generated, checked, and incorporated into the semantic model. The resulting labels serve as an initialization mechanism: Users remain responsible for validating and refining them within PLASMA's interface (see figure C.1).

Problem Statement. The construction of a semantic model SM requires the creation of semantic labels that ground dataset elements in a KG. In our formalization (see section 5.2.1) this means assigning for relevant selectors $k \in \mathcal{K}$ (column names, JSONPaths, XPaths, ...), schema-level labels of the form

$$SL \;=\; \{\langle c(k,\star),\; p,\; t\rangle\},$$

together with identifier semantics (subject construction) and, where needed, explicit instance-level refinements on $c(k,v)$. Here, p fixes how observations under k are connected in the induced RDF graph, and t specifies the target constraint (data type in $\mathcal{I}_{DT}$ or entity/class constraint in $\mathcal{I}_E$).

The empirical experiments in this section focus on *semantic type detection* as a core subproblem of automated semantic labeling, i.e., predicting the constraint component t for attribute selectors. Concretely, we consider a typing function

$$\tau : \mathcal{K}_s \rightharpoonup \mathcal{I}_C \cup \mathcal{I}_{DT}, \qquad \tau(k) = t,$$

where $\tau(k) = t$ classifies what the values under selector k represent. While semantic-label tuples additionally capture the predicate choice p and the subject construction mechanism required for a complete mapping, the experiments below evaluate the extent to which LLMs can recover $\tau(k)$ (hence the target concept t in $\langle c(k, \star), p, t \rangle$) from dataset documentation and representative samples, providing an initialization that users can validate and refine in PLASMA.

For evaluation, let τ_{true} denote the reference constraint (from the corpus' semantic models) for each labeled selector k in a dataset, and let $\tau_{pred}(k)$ be the prediction produced by the labeling component. We measure accuracy per selector,

$$\text{acc}(k) = \begin{cases} 1 & \text{if } \tau_{pred}(k) \neq \tau_{true}(k), \\ 0 & \text{otherwise,} \end{cases}$$

and aggregate over the labeled selectors of the dataset (and across datasets) to assess prediction quality.

Experiments on LLM-based semantic labeling The following experiments are based on our publication [Hoseini et al., 2024a]. While the conceptual framework and analysis were developed jointly, A. Burgdorf was primarily responsible for the experimental design and execution.

To justify using LLMs to automate semantic labeling we provide an empirical foundation based on four experiments in realistic industrial settings. For these experiments, we employed three advanced state-of-the-art LLMs:
(1) GPT-4, (2) GPT-4o, and (3) LLaMA 3-70B.

For evaluation, we manually selected three datasets from the **VC-SLAM corpus**[8] [Burgdorf et al., 2022] which contains datasets created by human modelers in combination with their data description and semantic model as evaluation datasets:

- Dataset 1 (VC-SLAM 0001) has seven labels that are close to natural language,
- Dataset 2 (VC-SLAM 0018) has 21 labels that are mostly human-readable, and
- Dataset 3 (VC-SLAM 0068) consists of 24 labels, some of which are abbreviations.

The experiments are briefly described in the following and the results are given in Table 5.4. Accuracy is computed as the proportion of dataset labels for which the model selected the correct ontology concept, as defined in the ground truth annotations of the VC-SLAM corpus.
Experiment 1 - Mapping to VC-SLAM: This experiment explores the ability of LLMs to map dataset labels to the corresponding concepts within the VC-SLAM ontology, provided solely in Turtle (TTL) format, without any additional contextual information. This setup is designed to evaluate the basic capability of LLMs to leverage the ontology's structure and content for semantic type detection. The task involves providing dataset labels to the LLM and instructing it to identify the most appropriate ontology concept for each label (see section 5.7.2).
Experiment 2 - VC-SLAM with Documentation: In this experiment, the methodology is similar to the *Mapping to VC-SLAM* experiment, but it includes comprehensive documentation of the VC-SLAM ontology and datasets. This tests the hypothesis that additional contextual infor-

[8] https://github.com/tmdt-buw/vc-slam, accessed on 02.05.2026

Prompt for Experiment 1

- You are a tool for semantic type detection. I will provide you an owl ontology that consists of all the concepts you know. This ontology is called VC-SLAM. Later I will additionally provide the labels of three data sets. For each label you return the fitting concept from the ontology.
- The first data set consists of the following labels: type, longitude, address, latitude, tvm_identifier, pay_by_credit_card, pay_by_cash
  Please return the results in the following form: label, concept

		ds1	**ds2**	**ds3**
	Nr. of labels	7	21	23
Total Hits (Accuracy) GPT-4	Mapping to VC-SLAM	4 (0.57)	7 (0.33)	3 (0.13)
	VC-SLAM with Documentation	5 (0.71)	13 (0.62)	5 (0.22)
	schema.org Ontology	7 (1.0)	16 (0.76)	11 (0.48)
	Simplified VC-SLAM	4 (0.57)	9 (0.43)	12 (0.52)
Total Hits (Accuracy) GPT-4o	Mapping to VC-SLAM	5 (0.71)	21 (1.0)	6 (0.26)
	VC-SLAM with Documentation	5 (0.71)	19 (0.9)	23 (1.0)
	schema.org Ontology	7 (1.0)	18 (0.86)	20 (0.87)
	Simplified VC-SLAM	5 (0.71)	16 (0.76)	21 (0.91)
Total Hits (Accuracy) Llama3 70b	Mapping to VC-SLAM	4 (0.57)	11 (0.52)	13 (0.57)
	VC-SLAM with Documentation	4 (0.57)	8 (0.38)	7 (0.3)
	schema.org Ontology	7 (1.0)	14 (0.67)	18 (0.78)
	Simplified VC-SLAM	2 (0.29)	6 (0.29)	1 (0.04)

Table 5.4: The table shows the results of the four experiments that we run on three datasets. The datasets refer to the following original datasets from VC-SLAM [Burgdorf et al., 2022]: 1: 0001, 2: 0018, 3: 0068

mation enhances the accuracy of semantic type detection.

Experiment 3 - schema.org Ontology: This experiment shifts the focus to a general-purpose ontology to compare the adaptability and performance of the LLMs with a different ontology structure. It provides insights into the model's versatility and the challenges of applying a broad ontology like *schema.org* to a specific dataset, highlighting differences in specificity and applicability. Results are manually compared with the described concepts in the data documentation.

Experiment 4 - Simplified VC-SLAM: The final experiment aims to investigate the impact of ontology complexity on semantic type detection accuracy. By reducing the VC-SLAM ontology to only include concept names and their descriptions without further relations, this experiment seeks to determine whether a simplified ontology framework would enhance mapping accuracy of LLMs due to decreased complexity and ambiguity.

Results

The outcomes of the experiments are measured across the three different datasets:
Experiment 1 - Mapping to VC-SLAM: In this experiment, the models mapped the labels directly to the VC-SLAM ontology. GPT-4 achieved 57% accuracy for ds1, 33% for ds2, and a notably lower 13% for ds3, indicating its struggle without additional contextual data. GPT-4o showed improved performance with 71% for ds1, 100% for ds2, and 26% for ds3, demonstrating better handling of the VC-SLAM ontology, especially for the second dataset. Llama3 70b had 57% for ds1, 38% for ds2, and 13% for ds3, performing similarly to GPT-4, which suggests that it also struggles without additional context.
Experiment 2 - VC-SLAM with Documentation: When provided with additional documentation, the models showed improved performance. GPT-4 achieved 71% accuracy for ds1, 61% for ds2, and 21% for ds3. GPT-4o further improved with 71% for ds1, 71% for ds2, and 43% for ds3. Llama3 70b had 57% for ds1, 38% for ds2, and 30% for ds3. These findings highlight the significant impact of extra contextual information in enhancing semantic type detection capabilities, with GPT-4o benefiting the most from the documentation.
Experiment 3 - schema.org Ontology: This experiment demonstrated the adaptability of the models with the schema.org ontology. GPT-4 showed strong performance with 100% accuracy for ds1, 76% for ds2, and 47% for ds3. GPT-4o performed exceptionally well, achieving 100% for ds1, 85% for ds2, and 87% for ds3. Llama3 70b also showed impressive results with 100% for ds1, 67% for ds2, and 78% for ds3. These results suggest that all models, particularly GPT-4o, handle general-purpose ontologies like schema.org effectively, possibly due to their inclusion in the training data or the meaningfulness of the descriptions.
Experiment 4 - Simplified VC-SLAM: In this experiment, the models mapped labels to a simplified version of the VC-SLAM ontology. In comparison to experiment 1 we achieved the following results: GPT-4 achieved same results for ds1, improvements of 10% ds2, and 39% for ds3, indicating that simplification partly improves performance. GPT-4o showed same results for ds1, a performance decrease of 24% for ds2 and an improvement of 65%. In contrast, Llama3-70b had performance losses of 28% for ds1, 23% for ds2, and 53% for ds3, indicating a significant struggle with the simplified ontology compared to the other models.

During these experiments, several key findings emerged. First, the availability of additional contexts, such as ontology documentation, significantly improves the models' ability to accurately map dataset labels to ontology concepts, underscoring the importance of rich contextual information for semantic type detection. Second, the experiments revealed the models' adaptability to different ontologies, with performance variations highlighting their capability to handle both specialized and general-purpose ontologies. Notably, the schema.org ontology showed high accuracy, particularly for GPT-4o. Lastly, the simplification of the ontology structure was shown to potentially enhance semantic type detection accuracy, particularly benefiting GPT-4o, although this was not a universal trend. These findings contribute valuable insights into the potential of leveraging LLMs for semantic type detection, indicating promising pathways for automating and refining the data categorization process. The experiments underscore the

significance of ontology design and contextual information in optimizing the performance of semantic type detection tasks using AI models like GPT-4 and GPT-4o.

These findings provide empirical justification for integrating LLMs into semantic data management workflows in SEDAR, particularly in automating semantic type detection as part of metadata generation (see section 4.3.1).

5.7.3 Semantic Modeling Workflow in SEDAR

The evaluation of semantic modeling platforms in table 5.3 identified PLASMA as the most suitable foundation for extending the SDM capabilities of SEDAR. PLASMA provides a mature, ontology-driven modeling workflow that aligns well with SEDAR's requirements: support for ontology evolution, user-guided refinement, and the ability to incorporate heterogeneous datasets into a unified semantic representation.

From an architectural perspective, PLASMA is treated as an external semantic modeling engine. Instead of integrating its codebase directly, its backend is implemented in Java and its frontend in Angular, SEDAR interacts with PLASMA through a well-defined API boundary. This preserves PLASMA's modularity and avoids technology conflicts within SEDAR's Python/React-based architecture, while still enabling deep integration of semantic models into SEDAR's metadata and querying pipeline.

PLAMSA's internal services (schema analysis, data modeling, knowledge graph services, etc.) are orchestrated as independent components. SEDAR communicates with these services to create, manage, and retrieve semantic models. A dedicated metadata class, *Modeling*, captures the semantic model's identity, associated datasets, and its syntactic representation. This ensures that semantic models, although computed in PLASMA, become first-class citizens within SEDAR's Workspace abstraction.

Semantic modeling in SEDAR begins after datasets have been ingested and enriched with initial metadata, including primary keys and preliminary semantic labels. Users may then initiate the creation of a semantic model through the *Semantic Data Management* tab. SEDAR delegates schema analysis and model initialization to PLASMA, but manages the orchestration and contextualization of datasets across the workspace.

The conceptual workflow is illustrated in figure 5.6. PLASMA generates an initial syntactic model based on sample data, while SEDAR contributes workspace context, selected ontologies, and dataset-level metadata. Together, these components enable a staged modeling process in which:

- schema structures are analyzed and represented,
- dataset-specific semantic models are generated,
- semantic connections across datasets are established, and
- the user iteratively refines the resulting model.

A key architectural extension concerns the ability to create semantic models spanning multiple datasets. PLASMA originally supported only single-source modeling; SEDAR augments this by introducing a unifying root entity that anchors multiple dataset models within a shared

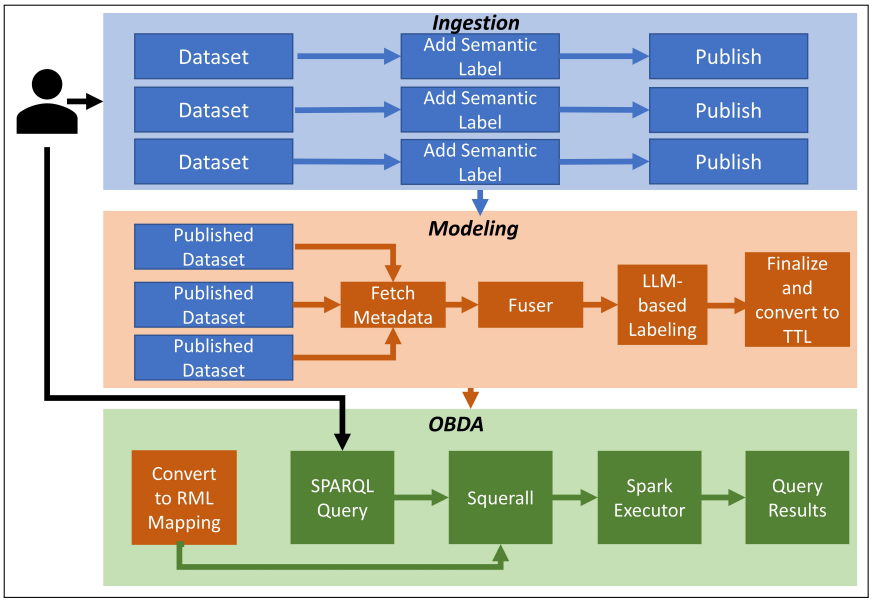


Figure 5.6: Semantic Data Processing pipeline

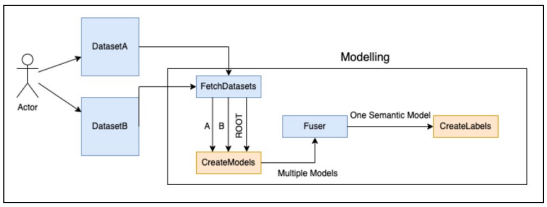


Figure 5.7: Generating one semantic model for multiple datasets

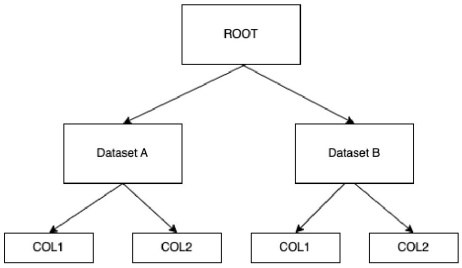


Figure 5.8: Combining multiple datasets

semantic space (see figures 5.7 and 5.8). Once individual dataset models are created, SEDAR fuses them into a consolidated model and returns it to PLASMA for user-guided refinement (see figure C.1). Users can edit classes, properties, and relationships directly within PLASMA's modeling environment, while SEDAR manages persistence, synchronization, and re-integration of the final model into the workspace.

5.7.4 Ontology-based Data Access in SEDAR

The final component of figure 5.6 operationalizes ontology-based data access in SEDAR by transforming semantic models into executable mappings and enabling SPARQL queries over heterogeneous datasets. This enables SEDAR to treat diverse storage technologies as a single, semantically unified information space. Fig. 5.4 sketches the query processing in SEDAR. Users formulate queries at the conceptual level using the ontology rather than the physical structure of sources, thereby decoupling data access from storage heterogeneity. Mappings are expressed in RML to link data sources to the KG. SEDAR's query engine translates the SPARQL queries using the mappings into queries to the underlying database systems and in-

```
plsm:X-rownumber
  ---id:<dataset_id1>_1
  rdf:type
      dbo:Capital> ;
  plcm:originalLabel
      "rowNumber
       ---id:<dataset_id1>" .

plsm:X-col1_1  rdf:type
    dbo:Country> ;
  plcm:originalLabel  "col1" .

plsm:X-col2_1  rdf:type
    dbo:City> ;
  plcm:originalLabel  "col2" .
```

Listing 5.1: Example mappings extract from PLASMA for the Captials dataset

```
<1> a rr:TriplesMap ;
  rml:logicalSource [
      nosql:store nosql:json ;
      rml:source "<dataset_id1>"
          ] ;
    rr:predicateObjectMap [
        rr:objectMap [
        rml:reference "col2" ] ;
    rr:predicate dbo:City ],
    [ rr:objectMap [
        rml:reference "col1" ] ;
    rr:predicate dbo:Country ] ;
    rr:subjectMap [ rr:class
        dbo:Capital ;
    rr:template "rowNumber" ] .
```

Listing 5.2: RML mapping for the Captials dataset

tegrates the results. In this way, SEDAR allows querying heterogeneous data sources using a uniform semantic data model.

RML mappings SEDAR's OBDA query engine expects RML mappings together with a corresponding SPARQL query as inputs (see section 5.2). To enable ontology-based data access, a finalized semantic model, must therefore be serialized into an executable mapping representation. This serialization is realized by transforming semantic labels (c_i, p, o) into RML predicate–object mappings that bind ontology predicates and classes to concrete dataset attributes. The PLASMA component provides the semantic structure of the model, materializing semantic labels and typing relations as RDF triples annotated with their originating data elements. SEDAR then extracts the relevant typing and relational triples from this semantic model and compiles them into RML `TriplesMap` definitions (see listings 5.1 and 5.2). In this process, data elements $c_i = (k_i, v_i)$ are translated into `rml:reference` expressions, while predicates and classes from the knowledge graph define the corresponding `rr:predicate`, `rr:class`, and `rr:subjectMap` components. All generated `TriplesMap` objects are stored as reusable mapping artifacts in SEDAR's metadata store. In particular, the `plcm:originalLabel` annotations in the PLASMA model correspond to the data elements c_i of semantic labels, which are mapped to `rml:reference` expressions in the resulting RML `TriplesMap`.

Squerall SEDAR's OBDA engine is based on the implementation of M. Mami *Squerall*, which stands for *Semantically Query All* [Mami et al., 2019a]. The original implementation[9] was written in Scala and included a Spark execution module. Within SEDAR, the application is compiled into a binary file (.jar) and executed through a bash command. The external package is compiled in Scala as it relies heavily on Apache Jena, for which no equivalent, mature Python alternative exists. SEDAR integrates a streamlined version of Squerall tailored to its metadata model: source discovery is driven directly from the mapping definitions, and all data access is executed through the `get_source_df` abstraction. The integrated Squerall engine performs three essential tasks:

[9] https://github.com/EIS-Bonn/Squerall/tree/master, accessed on 26.02.2025

1. query analysis to extract triple patterns and projection variables,
2. logical planning to determine join orders and pushdown opportunities, and
3. source mapping to identify physical datasets relevant to each triple.

SEDAR reuses these components to produce a unified logical query plan that can be executed over Spark.

Spark Executor To execute an OBDA task, SEDAR forwards the selected mapping and SPARQL query to the embedded Squerall executor, which generates a logical plan and executes it over Spark. SEDAR's OBDA executor receives the logical plan from Squerall and materializes it as Spark DataFrames. It loads the required dataset fragments, applies filters and transformations inferred from the mappings, performs semantic joins, and assembles the final result according to the SPARQL query semantics (see Algorithm 2). The executor implements

Algorithm 2: OBDA Query Processing Workflow

```
Input: workspace_id, mapping_id, query_string
Output: Query result
Retrieve the mapping by mapping_id
Prepare the mapping and query for processing
Parse the mapping file and the query using the OBDA engine;
Process the query results;
Inside result processing:
    foreach result do
        Load the corresponding data source;
        Apply required transformations and filters;
    if joins exist then
        Perform joins;
    if grouping, ordering, duplicate elimination, or limiting is required then
        Apply the respective operations;
```

the standard SPARQL algebra—joins, projections, grouping, ordering, and filtering—directly over Spark. This enables SEDAR to evaluate semantic queries across disparate storage backends using a unified execution model.

OBDA tab SEDAR provides an interactive OBDA interface that exposes the conceptual query model to end users. In the frontend, users interact with this functionality through the *OBDA* tab, where they can select, edit, or define mappings. Once a mapping is selected, the query text box shows up allowing to enter a SPARQL query. The query and selected mapping are forwarded to SEDAR's OBDA engine, which evaluates the request using the workflow described above.

In figure 5.9 this process is illustrated. In the example, we illustrate how the combined semantic model allows querying the three individual datasets (Country, Capitals and Currency) in a joined representation. The example illustrates Squerall's pattern-based query syntax, where dataset instances are bound to indices, predicates identify attributes, and joins are expressed through shared semantic relationships. The query logic follows the Squerall framework, in which one first defines the indices of the datasets, e.g. `?country_index rdf:type dbo:country`. Then, one defines individual columns for selection, e.g. `?capital_index dbo:City`

`?capital`. Finally, if joining is necessary, one defines column pairs for joining, e.g. `?country_index dbo:country ?currency_index` and `?currency_index dbo:country ?country_index`. This syntax is translated into queries for the underlying databases. In this example, which was presented in [Hoseini et al., 2023], each dataset has been ingested into a different storage database: the *Country* dataset remains in *HDFS*, the *Capital* dataset remains in *MongoDB*, and the *Currency* dataset remains in *PostgreSQL*. SEDAR is responsible for loading the datasets into Spark DataFrames and performing the joins inferred from the mappings.

5.8 Conclusion

This chapter addressed RQ3 by describing how semantic data management is implemented within the SEDAR system to improve the integration and usability of heterogeneous data in an industrial context.

Through a combination of ontology-driven modeling, knowledge graph integration, and LLM-based semantic type detection, SEDAR enhances the interpretability, interoperability, and reusability of diverse datasets. Section 5.3 demonstrated how LLMs can automate metadata enrichment and semantic labeling, while section 5.4 explored methods for unifying KGs with LLMs to assist in semantic model creation. Furthermore, section 5.7 detailed how these components are operationalized in SEDAR using platforms such as PLASMA and support OBDA.

By embedding semantics into the data lake architecture, SEDAR enables more meaningful data discovery, integration, and analysis especially for users who lack deep technical knowledge but require domain-aware tools to work with complex data. This semantic foundation supports the broader goal of bridging gaps between data producers, domain experts, and data scientists in industrial chemistry.

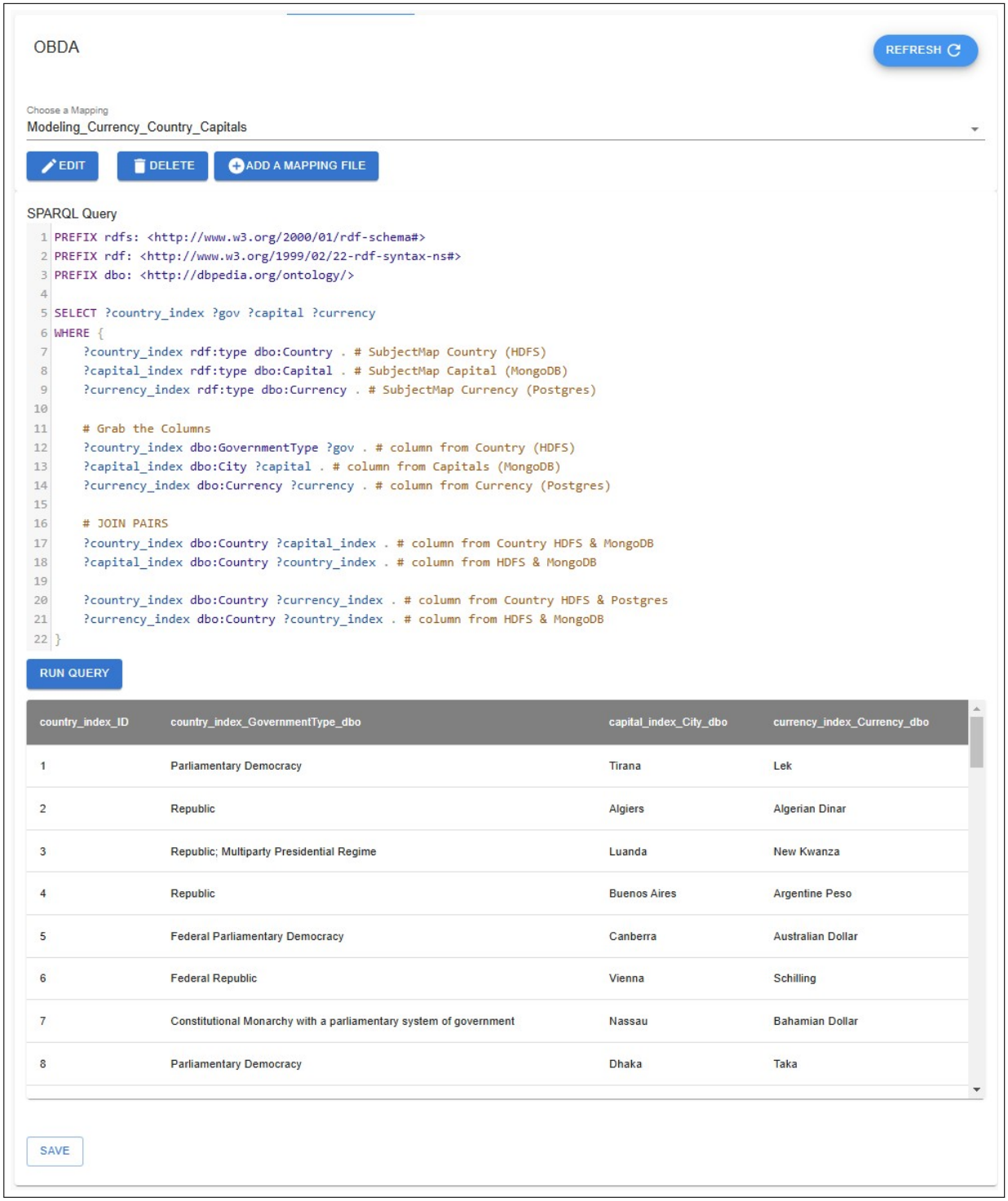


Figure 5.9: The *OBDA* tab displaying a SPARQL query to find currencies and capitals of countries.

6

Automating Machine Learning in SEDAR

"Intelligence is the ability to adapt to change."

Stephen Hawking

Modern data-intensive environments increasingly require automated, scalable, and explainable machine learning workflows. As industrial data ecosystems grow in complexity, manual model development becomes both a bottleneck and a source of inconsistency. Addressing RQ4, this chapter examines how SEDAR evolves from a semantic data platform into a system capable of supporting automated, reproducible, and intelligent ML workflows.

Our work proceeded in three stages, reflected in the literature on which this chapter builds. First, foundational MLOps concepts were integrated into SEDAR's architecture, establishing the operational backbone needed for reliable ML orchestration. Second, we developed an Automated Machine Learning module [Hoseini et al., 2024c] that automates model selection, hyperparameter tuning, evaluation, and deployment, thereby lowering the barrier for non-experts. Finally, our recent work explores how emerging LLMs and agent-based systems can transform data science workflows by providing flexible problem understanding, autonomous pipeline execution, and semantic alignment with domain knowledge [Hoseini et al., 2025a,b].

Together, these developments illustrate a progression from operational ML support toward intelligent, semantically grounded automation.
This chapter is largely based on the publications [Hoseini et al., 2024c, 2025b].

Chapter Outline

This chapter is structured as follows:

- Section 6.1 formulates the design goals and problem statement for integrating AutoML into industrial data platforms.
  - Section 6.1.1 presents a unified, backend-agnostic abstract logic for specifying AutoML experiments.
  - Section 6.1.2 presents the implementation of the *AutoMLWrapper* as a configuration-driven mediation layer over heterogeneous AutoML backends and LLM-based code generation to assist AutoML workflows in notebook environments.
- Section 6.2 introduces LLM-based data science agents and motivates their use in industrial analytics.
  - Section 6.2.1 reviews related work on agentic automation across the data science lifecycle.
  - Section 6.2.2 develops a neuro-symbolic architecture for data science agents.
  - Section 6.2.3 analyzes the role of knowledge graphs as structured long-term memory for grounding agent behavior.
  - Section 6.2.4 presents an implementation of a KG-enhanced data science agent using MLSea and AssistML.
- Section 6.3 concludes the chapter by assessing the contributions with respect to the research questions (section 1.2).

6.1 Conceptual Foundations of Automated ML in SEDAR

Modern data platforms are no longer limited to the ingestion, storage, and preparation of heterogeneous data, but must also support the development, execution, and operationalization of machine learning (ML) applications. In industrial environments, analytics workflows are expected to be reproducible, scalable, and tightly integrated with existing data infrastructures. As a consequence, data platforms increasingly adopt principles from MLOps to manage the lifecycle of ML artifacts, experiments, and deployments systematically.

While MLOps frameworks address important operational challenges, such as experiment tracking, reproducibility, and model versioning, they do not eliminate the fundamental complexity of developing ML applications. In practice, users are still required to manually design analytical workflows, select suitable algorithms, tune hyperparameters, and ensure that resulting models can be deployed and maintained reliably. This reliance on specialized expertise constitutes a major scalability bottleneck, as analytical demand grows faster than the availability of skilled data scientists.[1]

Automated Machine Learning (AutoML) has emerged as a response to this challenge by automating well-defined sub-tasks of the ML pipeline, most notably model selection and hyper-

[1] https://hbr.org/2015/05/data-scientists-dont-scale, accessed 20.01.2025

Definition: An AutoML experiment AE within SEDAR is described by the tuple

$$AE = (D, T, P, C, E, M),$$

where:

- D denotes the **data type** (tabular, image, text, time series),
- T denotes the **task type** (classification, regression, segmentation, forecasting),
- P denotes the **problem type** (binary, multiclass, multilabel),
- C denotes **execution constraints** such as runtime limits or resource availability,
- E denotes the **evaluation metrics** used to assess and compare model quality,
- M denotes the **execution backend**, i.e., the concrete AutoML framework used.

parameter optimization. However, AutoML systems do not fully replace human analysts. Tasks such as problem formulation, data understanding, feature definition, and the interpretation of results still require domain expertise. Consequently, AutoML should be understood not as a replacement for human decision-making, but as an assistive technology that reduces manual effort in constrained parts of the workflow [Karmaker et al., 2021].

From a systems perspective, an additional challenge arises from the heterogeneity of existing AutoML frameworks. Different libraries expose distinct APIs, support different data modalities and task types, and rely on incompatible configuration mechanisms. As a result, AutoML outputs are difficult to compare, reuse, or integrate into larger analytical workflows without substantial manual effort. This fragmentation limits the practical adoption of AutoML in production-grade data platforms. The design goal pursued in SEDAR is therefore not merely to integrate AutoML as an isolated capability, but to provide a unified, platform-level abstraction for automated machine learning. This abstraction must:

1. decouple user intent from concrete execution details,
2. support heterogeneous AutoML backends consistently, and
3. ensure that automated analytics workflows are systematically documented, reproducible, and semantically linked to datasets, users, and downstream applications.

To this end, SEDAR builds upon established MLOps concepts [Alla and Adari, 2021, Kreuzberger et al., 2022, Zaharia et al., 2018] for lifecycle management, while extending them with a higher-level abstraction that enables automated and reproducible ML workflows (section 4.3.2 and section 6.1.2). The central design principle is to treat automated machine learning not as a collection of library-specific scripts, but as a first-class analytical process embedded in the platform's metadata and execution model. The realization of this principle is explored in the following sections, beginning with a conceptual abstraction of AutoML workflows and culminating in their architectural and implementation-level realization.

6.1.1 AutoML Abstraction

SEDAR introduces an abstraction to unify automated machine learning that decouples user intent from concrete execution mechanisms. Rather than exposing algorithmic choices, hy-

perparameters, or library-specific APIs, AutoML workflows are specified through a small set of conceptual dimensions that are sufficient to characterize a machine learning problem independently of its technical realization.

This representation captures the analytical intent of a machine learning task while remaining agnostic to implementation-specific details. Multiple concrete executions may therefore correspond to the same abstract experiment description. The tuple deliberately abstracts from feature engineering and data preparation steps, which are either handled internally by AutoML frameworks or delegated to upstream components of SEDAR. The selection of dimensions (e.g., task and data type) is constrained by the capabilities of the three chosen AutoML frameworks and therefore does not exhaust the full design space, but is intended to cover as broad a range of practically relevant use cases as possible (section 6.1.2).

Decoupling User Intent from Execution. Users interact with SEDAR by specifying values for the abstract dimensions (D, T, P, C, E), without selecting concrete algorithms or tuning model-specific parameters. The execution backend M is resolved automatically by the system based on capability constraints and availability. This design shifts responsibility for algorithm selection, configuration, and orchestration from the user to the platform, enabling non-technical users to initiate complex machine learning workflows while preserving reproducibility and traceability.

Configuration Files as a Mapping Layer The AutoML abstractions needs to be grounded in concrete execution capabilities provided by heterogeneous AutoML frameworks. This grounding is realized through a configuration layer that encodes the capabilities, constraints, and parameter semantics of supported AutoML libraries. Conceptually, this layer constitutes an ontology-like schema over the AutoML domain, in the sense that it defines a shared vocabulary, admissible relations, and constraints over supported analytical capabilities. It defines:

1. which combinations of data types, task types, and problem types are supported,
2. which execution backends can realize a given abstract experiment description, and
3. how abstract configuration dimensions are translated into library-specific parameters.

Importantly, this semantic structure is not implemented using formal ontology languages, but instead, it is represented using structured configuration files (see next section).

From Abstract Configuration to Concrete Execution The transition from abstract configuration to executable machine learning workflows is realized through a controlled mapping process. For each supported AutoML library, the abstract configuration is translated into library-specific constructor calls, training routines, and parameter settings.

This mapping process is intentionally asymmetric: while the abstract configuration defines the minimal information required to execute an experiment, individual AutoML frameworks may introduce additional capabilities or constraints. SEDAR therefore enforces a common execution contract while allowing library-specific extensions when necessary. The resulting execution logic enables the system to dynamically select suitable AutoML frameworks, initialize experiments, and register results uniformly, without exposing these decisions to the user.

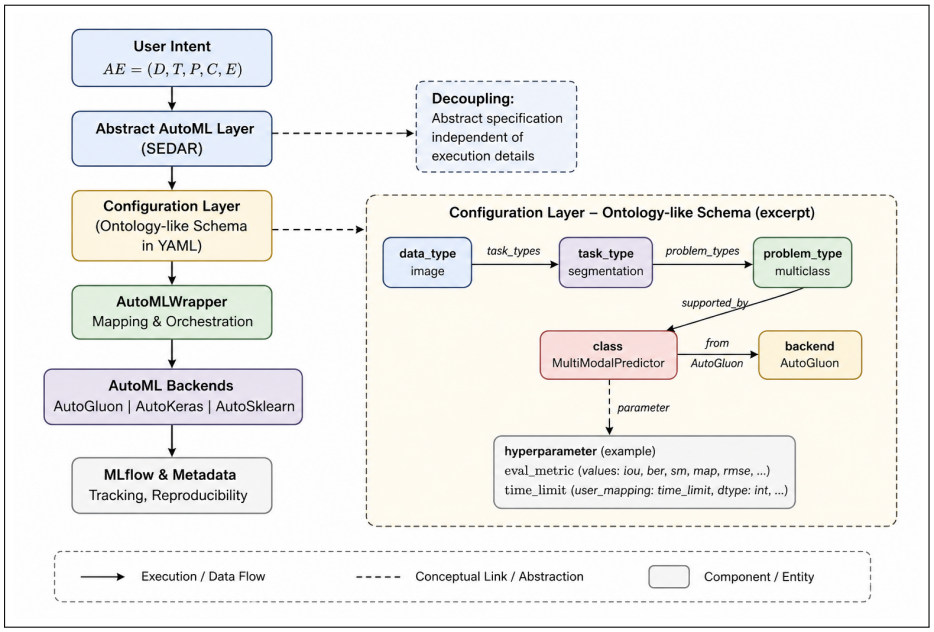


Figure 6.1: From abstract AutoML experiment specification to concrete execution in SEDAR. The configuration layer encodes an ontology-like schema reflecting the hierarchical structure of configuration files (data type, task type, problem type, and backend-specific classes) and associated parameter mappings.

The so-called **AutoMLWrapper** constitutes the operational realization of this ontology-like mapping layer. It interprets abstract experiment descriptions, resolves admissible execution backends, and translates the unified configuration into concrete library-specific calls while enforcing the constraints encoded in the configuration files. The architectural realization of this abstract logic, including its integration with execution environments, metadata management, and lifecycle tracking, is described in the following section.

6.1.2 Implementation

This section describes the architectural and technical realization of the unified AutoML abstraction introduced in section 6.1.1 and is illustrated in figure 6.1. The implementation builds upon established MLOps components to ensure reproducibility, traceability, and operational integration of automated machine learning workflows within SEDAR.

To manage the lifecycle of machine learning artifacts, SEDAR integrates selected components of MLFlow [Zaharia et al., 2018]. MLFlow provides a structured metadata model for tracking and reusing ML artifacts, organized around the following core abstractions:

- **Experiments** — containers grouping related machine learning runs.
- **Runs** — individual executions logging parameters, metrics, code versions, and artifacts.
- **Models** — trained artifacts including execution environment metadata.
- **Model Registry** — centralized management of model versions and lifecycle states.
- **Metrics** — task-dependent quality measures attached to runs.
- **Parameters and artifacts** — configuration values and serialized outputs stored in HDFS.

AutoMLWrapper

The AutoMLWrapper operationalizes the unified AutoML abstract logic introduced in section 6.1.1 by providing a concrete execution layer for automated machine learning within SEDAR. It serves as a mediation component between abstract experiment descriptions $AE = (D, T, P, C, E, M)$ and the heterogeneous APIs of existing AutoML frameworks, translating abstract experiment descriptions into executable workflows.

Concretely, the AutoMLWrapper is implemented as a Python module[2] that integrates multiple open-source AutoML systems, including *AutoKeras*, *AutoGluon*, and *AutoSklearn*. By standardizing configuration, execution, and result handling across these libraries, the AutoMLWrapper enables consistent experiment tracking, backend substitution, and reproducible analytics workflows without exposing library-specific complexity to the user.

Configuration-Driven Mediation of Abstract Experiments. To operationalize the abstract AutoML logic across heterogeneous AutoML frameworks, the AutoMLWrapper implements a configuration-driven mediation layer that (i) encodes framework capabilities and admissible problem specifications and (ii) maps the resulting abstract configuration to library-specific constructor/training calls and parameters. Concretely, this mediation relies on structured configuration files that capture supported combinations of data types, task types, and problem types as well as parameter metadata (e.g., types, ranges, and user-facing hints), while the wrapper normalizes common inputs through a shared representation and performs required modality-specific conversions when frameworks expect different formats.

Figure 6.2 illustrates this design: the global capability file (left) specifies which (D, T, P) combinations are supported by a given backend, the library-specific configuration (center) defines the relevant parameter space and mappings, and the code example (right) shows how an abstract specification (e.g., image segmentation with runtime constraints) is executed via the wrapper while remaining insulated from library-specific API details.

Metadata Management for Data Analytics

To operationalize the abstract AutoML experiment description, SEDAR extends its metadata model to explicitly represent each dimension of an automated machine learning workflow (see orange nodes in figure 4.6). This ensures that analytical intent, execution decisions, and empirical outcomes are persistently linked and can be inspected, reproduced, and compared across experiments.

The `AutoMLExperiment` entity captures the conceptual problem specification by recording the selected data type (D), task type (T), and problem type (P), thereby providing a stable, backend-agnostic representation of the analytical objective. The resolved execution backend (M), corresponding to the selected AutoML framework, is stored as part of this entity, making the translation from abstract configuration to concrete execution explicit and traceable.

[2] https://github.com/hsnr-data-science/SEDAR-ML, accessed on 02.06.2026

GlobalConf.yaml

```
AutoGluon:
 __classes:
  MultiModalPredictor:
   __data_types:
    image:
     __task_types:
      classification:
       __problem_types:
        - binary
        - multiclass
      few-shot-classification:
       ...
      zero-shot-classification:
       ...
      zero-shot-object-detection:
       ...
      ...
      segmentation:
       __problem_types:
        - binary
        - multiclass
```

AutoGluonConfig.yaml

```
AutoGluon:
 MultiModalPredictor:
  __hyperparameter_details:
   constructor:
    ...
   validation_metric:
    __has_values: true
     __values:
      - accuracy
      - f1
      - f1_macro
      - iou
      ...
 fit:
  time_limit:
   __has_values: false
    __dtype: int
    __user_mapping: runtime
    __hint: Time limit for the\
    experiment in seconds.
 ...
```

```
wrapper = AutoMLWrapper('autogluon')
# Enable parameters not present in config files
wrapper.AllowExtraHyperparameters(
    {'constructor': ['verbosity']}
    )

wrapper.Train(
    train_data=train_data,
    validation_data=val_data,
    target_column='label',
    data_type='image',
    task_type='segmentation',
    problem_type='binary',
    hyperparameters={
        'eval_metric':eval_metric,
        'validation_metric':eval_metric,
        'runtime':2700,
        'verbosity':4,
        ...
        }
)
```

Figure 6.2: Excerpts of the global (left), the *AutoGluon*-specific configuration file (center) and exemplary code of the AutoMLWrapper for image segmentation (right).

The `Notebook` entity represents execution context, which anchors an AutoML experiment to a concrete computational workspace and links it to dataset versions used during execution.

Execution constraints (C), such as runtime limits or resource-related parameters, are materialized at run level through MLFlow parameters and environment descriptors allowing subsequent experiments to be compared under equivalent conditions.

Evaluation outcomes (E) are captured through MLFlow metrics attached to individual runs and are linked back to the corresponding `AutoMLExperiment`. This enables systematic comparison of alternative executions of the same abstract experiment description, even when different AutoML backends or configurations are used.

For image-based datasets, additional lightweight metadata entities are introduced to capture structural aspects of the data. `ImageEntity` objects record image-level properties relevant to execution, while `FolderStructure` entities represent dataset organization required to interpret tasks such as classification, detection, or segmentation. These extensions ground the abstract data type (D) in concrete dataset characteristics without imposing assumptions on physical storage layouts.

LLM-based code generation for automating machine learning While the AutoML abstraction and its implementation provide a structured and reproducible mechanism for executing ML workflows, certain use cases require more flexibility than can be captured by predefined configurations. To address this, SEDAR additionally integrates LLM-based code generation as a complementary interaction and execution layer. This allows users not only to formulate AutoML experiments more easily, but also to generate and execute complete, customized machine learning pipelines directly within notebook environments.

In this setting, LLMs do not replace the AutoML abstraction, but extend it by enabling a second mode of operation: while the AutoMLWrapper enforces a controlled and comparable execution model, LLM-generated code provides a flexible alternative for exploratory or highly specialized workflows that fall outside the predefined configuration space. Both modes are in-

tegrated into the same execution environment and metadata infrastructure, ensuring that even LLM-generated workflows remain traceable, reproducible, and linked to datasets and experiments within SEDAR.

JupyterAI[3] is an extension for Jupyter that enables the use of LLMs within a separate chat window and the cells of a notebook. It distinguishes between language and embedding models. The embedding model is used to create a compressed representation of the contents of a particular directory, and the language model is used to ask specific questions about the file content. Users can either have methods or code from the notebook explained to them, or have code sections and entire notebooks generated. JupyterAI can be connected to a range of default models, but also to *Hugging Face* and *GPT4All*, which include a variety of open-source LLMs of varying sizes.

A new user interface based on *LangChain* has been implemented to enable communication between the user and LLM providers, allowing for the dynamic integration of additional information from the system's metadata. The current implementation offers a simple chat functionality from the frontend. The interface can be used with various LLM providers.

This interface alongside the JupyterAI extension will be used to generate code to craft ML models for various applications (see chapter 8) directly inside a notebook associated to with a specific dataset.

Integrating a JupyterHub Runtime Environment for AutoML For interactive development and execution, SEDAR integrates JupyterHub as its user-facing computational environment. Notebook templates are generated automatically based on the abstract AutoML configuration (e.g., task and data type) and provide boilerplate code for dataset loading, experiment initialization, run tracking, and model deployment.

By using an external JupyterHub server in combination with the container management software Kubernetes, user environments can be automatically managed, scaled, and monitored. Communication with the external JupyterHub now occurs exclusively via SEDAR's backend API. Both JupyterHub and user servers (Jupyter servers) implement their own APIs for different purposes. The JupyterHub API is used to start and stop a user's Jupyter server. The Jupyter server API is then used to upload and modify the content of Jupyter notebooks. Authentication between SEDAR, JupyterHub, and auxiliary services is unified through OAuth and JWT-based identity propagation (see section 4.3.4). This ensures that notebooks, datasets, and machine learning runs are consistently associated with the correct users and workspaces. For minimal privilege operation within the external Jupyter installation, a user-specific API token is used for each Jupyter server. A custom-designed Docker image for SEDAR comes with the Jupyter installation, AutoML environment, and GPU support.

The various presented modules are evaluated in detail on use case from our research domain situated in data-driven chemistry in section 8.5.

[3] https://jupyter-ai.readthedocs.io/en/latest/, accessed on 02.05.2026

6.2 Data Science Agents

LLM-based agents [Xi et al., 2025] excel at natural language understanding and reasoning with goal-oriented behaviors, tool use, and autonomous control flow. Rather than producing a single response, agents are designed to iteratively plan, act, observe intermediate results, and adapt their behavior by invoking external functions, APIs, or execution environments. Recent frameworks [Patil et al., 2024, Wang et al., 2024, Hollmann et al., 2024, Hong et al., 2024] demonstrate that such agentic patterns can be successfully applied to data-intensive tasks, including data transformation, exploratory analysis, and machine learning pipeline orchestration.

The growing interest in autonomous agents is partly motivated by the observation that continued performance gains from scaling data and pre-training are approaching practical limits. At NeurIPS 2024, Sutskever argued that while compute continues to scale, the availability of high-quality training data has largely plateaued, shifting emphasis toward systems that can reason, plan, and act autonomously [Sutskever, 2024]. In the context of data analytics, this has led to the emergence of *data science agents*: LLM-based systems capable of translating high-level analytical goals into executable workflows, monitoring their progress, and iteratively refining results. Conceptually, these agents delegate complex analytical subtasks to specialized components, effectively using a powerful general-purpose model to orchestrate smaller, task-specific processes.

Despite their promise, current data science agents face fundamental challenges when deployed in complex, industrial data environments. They must (i) reason over heterogeneous data modalities, (ii) plan multistep workflows with strong inter-step dependencies, (iii) integrate domain knowledge beyond textual context, and (iv) remain controllable, reproducible, and explainable. The remainder of this section reviews the state of the art, identifies these limitations more precisely, and motivates the integration of symbolic knowledge in the form of knowledge graphs as a core design principle.

6.2.1 Related Work

We present existing work on data science agents as an evolution in three broad phases: (i) LLM-assisted AutoML components, (ii) end-to-end agentic data science systems, and (iii) experience- and memory-driven agents.

LLM-Assisted AutoML Components Early work focuses on integrating LLMs into specific stages of the AutoML pipeline. Tornede et al. [2023] survey bidirectional interactions between AutoML and LLMs, highlighting improvements in usability, configuration assistance, and result interpretation, while also warning of risks such as hallucinated configurations and uncontrolled resource usage [Ji et al., 2023]. Several approaches employ LLMs for hyperparameter optimization or architecture search [Zhang et al., 2023c,b, Liu et al., 2024c, Shen et al., 2024, Chen et al., 2023a], or for automating feature engineering [Hollmann et al., 2024, Li et al., 2025, Malberg et al., 2024]. While effective for isolated subtasks, these methods do not address the

coordination and planning of complete analytical workflows.

End-to-End Data Science Agents A second line of work investigates agents capable of orchestrating entire data science pipelines. A key challenge is the strong interdependence between pipeline stages: dataset characteristics influence preprocessing choices, model architectures, and hyperparameter spaces, which in turn affect evaluation and deployment strategies [Trirat et al., 2024, Hong et al., 2024]. Systems such as the *Data Interpreter* [Hong et al., 2024] construct explicit task graphs to capture these dependencies and generate executable action graphs for stages such as exploration, feature engineering, training, and evaluation, with iterative debugging loops driven by execution feedback.

Other frameworks emphasize multi-agent collaboration. *AutoML-Agent* [Trirat et al., 2024] decomposes the workflow into specialized agents for prompts, data, models, and operations, while enforcing multi-stage verification. *DS-Agent* [Guo et al., 2024] integrates Case-Based Reasoning to reuse successful historical solutions, reducing reliance on the raw generative capacity of LLMs.

Experience- and Memory-Driven Agents Recent systems such as *AgentK* [Grosnit et al., 2024] emphasize long-term memory and experience reuse. By storing and reasoning over past executions, these agents generalize across tasks and domains and achieve competitive performance in realistic benchmarks such as Kaggle competitions. However, experience is typically stored in unstructured form, limiting transparency and principled reuse.

Summary Overall, LLM-based agents outperform traditional AutoML systems on complex, multistep analytical tasks but remain brittle in the absence of explicit structure. They are prone to hallucinated code, inconsistent planning, and opaque decision-making, with performance strongly dependent on the underlying LLM. These limitations motivate the integration of explicit, symbolic representations to constrain, guide, and explain agent behavior.

6.2.2 Neuro-Symbolic Data Science Agents

Recent surveys on LLM-based data science agents [Sun et al., 2025, Bhuyan et al., 2024] describe a commonly adopted architecture in which an LLM serves as the central control component, supported by mechanisms for planning, reasoning, reflection, and error handling, and coupled to a sandboxed execution environment for safe code execution.

From a systems perspective, however, such architectures remain predominantly *neural*. Task structure, execution constraints, domain knowledge, and prior analytical experience are encoded implicitly in prompts or transient context windows. As a result, agent behavior is difficult to constrain, reproduce, or systematically improve over time. These limitations become particularly pronounced in industrial settings where analytical workflows must be explainable, comparable across runs, and grounded in durable knowledge.

To address these challenges, we conceptualize data science agents as *neuro-symbolic*

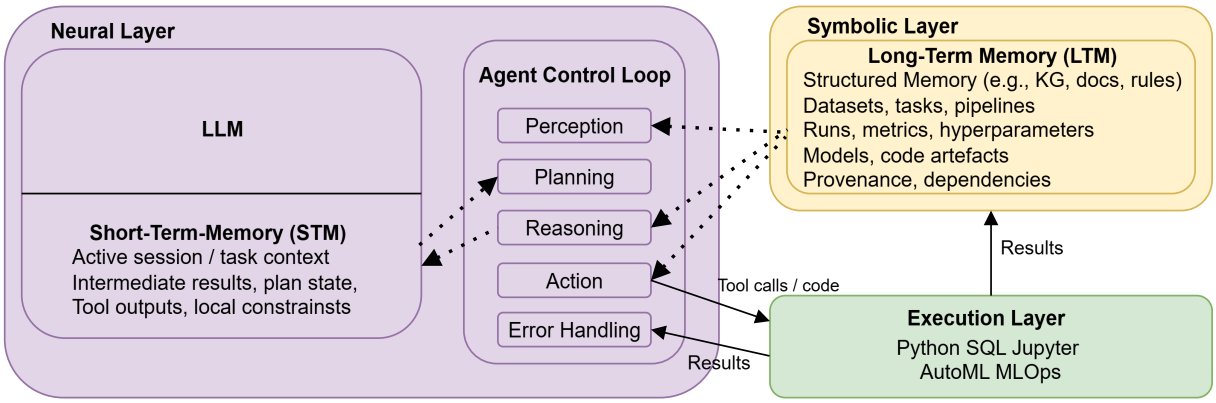


Figure 6.3: Neuro-symbolic architecture of a data science agent, illustrating the interaction between an LLM-based agent control loop, short- and long-term memory, and the execution environment. Dotted arrows denote neural–symbolic interactions. Solid arrows denote control and execution flows. Inspired by [Sun et al., 2025, Bhuyan et al., 2024]

systems. Rather than treating the LLM as a monolithic problem solver, agent behaviour is decomposed into an explicit functional loop supported by structured memory and a controlled execution environment. Figure 6.3 illustrates this layered architecture and the interaction between neural and symbolic components.

Agent Control Loop. User interaction, perception, planning, reasoning, and reflection form an explicit *agent control loop*. This loop captures the cognitive responsibilities typically attributed to LLM-based agents. Perception aggregates user instructions, intermediate results, dataset characteristics, and retrieved knowledge into a coherent task representation. Reasoning operates on this representation to validate assumptions and derive decisions, while planning constructs a sequence (or graph) of analytical actions. Action translates plans into concrete tool invocations, and reflection and error handling incorporate execution feedback, enabling plan revision and autonomous debugging. The LLM constitutes the neural substrate that realizes this loop. Its strength lies in flexible language understanding, abstraction, and generation; however, without explicit external structure, the loop alone cannot guarantee grounding, consistency, or long-term learning across tasks.

Memory as the Neuro-Symbolic Interface. A defining characteristic of the neuro-symbolic architecture is the explicit treatment of memory as a first-class architectural element. As illustrated in Figure 6.3, we distinguish between short-term memory (STM) and long-term memory (LTM), each serving a complementary role.

Short-term memory is ephemeral and task-local. It maintains the active session context, including conversational state, intermediate artifacts, partial plans, and recent tool outputs. STM stabilizes multi-step reasoning and planning by ensuring coherence across iterations of the agent control loop.

Long-term memory is persistent and reusable, in contrast to the transient, context-bound state of short-term reasoning. It stores structured symbolic knowledge such as datasets, tasks,

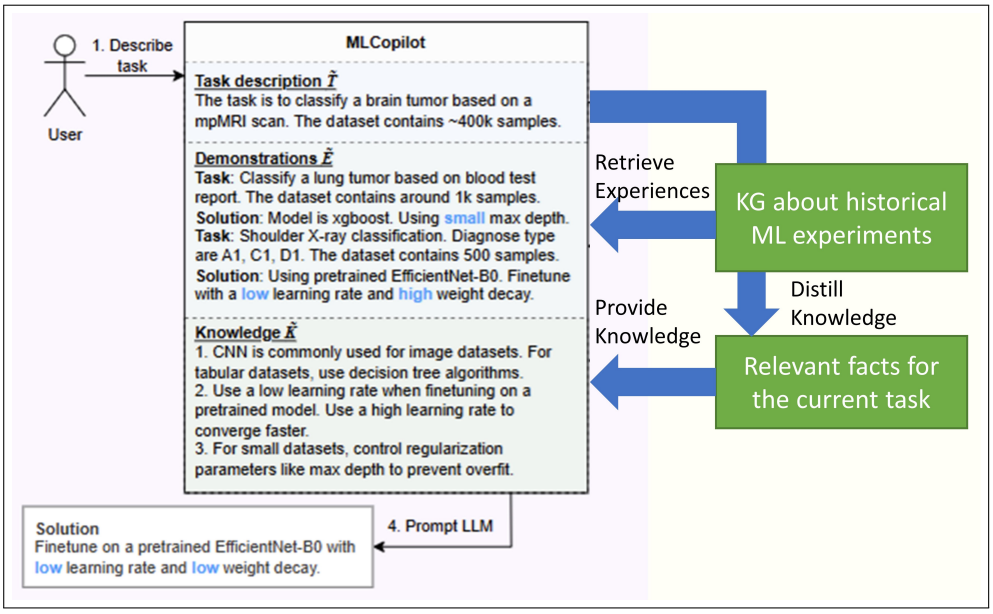


Figure 6.4: A KG-enhanced data science agent, extended from Zhang et al. [2023b].

pipelines, historical runs, metrics, models, code artifacts, and provenance information. By externalizing this knowledge into durable representations, the agent gains access to grounding, constraint-aware reasoning, and systematic reuse of prior analytical experience. The realization of such long-term memory through the systematic integration of large language models with knowledge graphs is detailed in the following sections.

Execution Environment and Action. Action is realized through a sandboxed execution environment, shown in the lower part of Figure 6.3. This environment mediates all interactions with external tools, including notebooks, databases, AutoML frameworks, and MLOps services.

6.2.3 Integrating Knowledge Graphs with Data Science Agents

A central limitation of purely neural data science agents is their reliance on implicit knowledge encoded in model parameters or transient prompt context. To partially mitigate this, post-training enhancement mechanisms such as Retrieval-Augmented Generation (RAG) enrich LLMs with external information retrieved at inference time. In data science settings, this typically corresponds to retrieving prior solutions, code snippets, documentation, or experimental outcomes, as demonstrated in systems such as *DS-Agent*, *AutoML-Agent*, and *AgentK* [Guo et al., 2024, Trirat et al., 2024, Grosnit et al., 2024].

While effective in extending factual coverage, traditional RAG approaches treat retrieved knowledge primarily as unstructured text. As shown by Peng et al. [2024], this neglects explicit relational structure, introduces redundancy, and limits an agent's ability to reason globally across interconnected resources. As a consequence, important analytical dependencies, such as relationships between datasets, tasks, models, metrics, and constraints, remain implicit and

difficult to validate or reuse.

Knowledge graphs address these limitations by representing domain knowledge as explicit entities and relations. They enable precise retrieval, constraint-aware reasoning, provenance tracking, and explainable decision-making through structured queries. However, knowledge graphs are costly to construct and maintain, evolve slowly compared to textual corpora, and lack the generative flexibility required to interpret ambiguous user intent or synthesize novel analytical workflows [Iglesias et al., 2023].

From a systems perspective, retrieval-augmented agents rely on some form of search mechanism over unstructured text, semi-structured documents, or explicitly structured knowledge. Viewed through this lens, search engines, knowledge graphs, and LLMs represent complementary rather than competing technologies. Search engines provide broad and up-to-date coverage but limited structural semantics; knowledge graphs offer precise, curated, and explainable representations but restricted coverage; and LLMs excel at abstraction, reasoning, and generation while suffering from hallucinations and limited factual grounding. Table 6.1 summarizes these trade-offs.

Viewed together, search engines, knowledge graphs, and LLMs represent complementary rather than competing technologies. Search engines provide broad and up-to-date coverage but limited structure; knowledge graphs offer precise, curated, and explainable representations but restricted coverage; and LLMs excel at abstraction, reasoning, and generation while suffering from hallucinations and limited factual grounding. Table 6.1 summarizes these trade-offs.

From a neuro-symbolic perspective, these observations motivate agent architectures in which neural reasoning is systematically grounded in symbolic memory. KG-enhanced LLMs [Hu et al., 2023] integrate structured knowledge to improve factual consistency, controllability, and interpretability, while experience-driven systems such as *CoML* [Zhang et al., 2023b] demonstrate how historical analytical outcomes can guide future decisions. Figure 6.4 illustrates this principle by extending experience-based data science agents with a knowledge graph that stores structured information about datasets, tasks, models, and results.

In the context of data science agents, knowledge graphs therefore emerge as a principled realization of long-term symbolic memory. They externalize analytical knowledge that would otherwise remain implicit in prompts or execution traces, enabling agents to ground perception, constrain planning, and contextualize action using durable, reusable representations. The following section builds on this conceptual foundation and describes how such symbolic memory can be instantiated and integrated into the execution architecture of data science agents.

6.2.4 Implementation of a KG-enhanced Data Science Agent

This section presents an implementation of a knowledge-graph-enhanced data science agent that instantiates the *symbolic layer* of the neuro-symbolic architecture introduced in the previous section. Specifically, it realizes *long-term memory* as an explicit, structured knowledge graph capturing historical machine learning experience, and integrates this memory into the agent's perception and planning loop through controlled retrieval mechanisms.

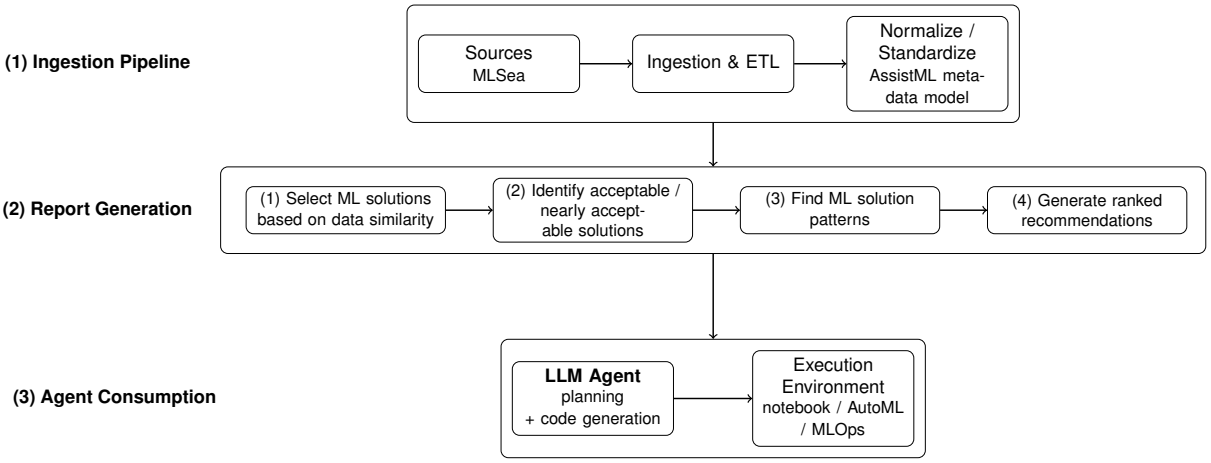


Figure 6.5: Workflow for instantiating long-term symbolic memory in a data science agent.

Architectural Scope

Within the neuro-symbolic agent architecture, the implementation described in this section addresses three core aspects:

1. *symbolic long-term memory*, realized by the **ingestion** of a knowledge graph over datasets, tasks, models, and experiments into a structured metadata repository;
2. *grounded perception*, in the form of a **report** providing the agent with historical context for a given analytical task; and
3. *planning support*, as the **agents consumes** prior solutions, configurations and performance outcomes as reusable knowledge and integrates them with its internal short-term memory to generate code. The execution control remains user-controlled in a sandboxed execution environment.

Figure 6.5 illustrates the concrete workflow by which long-term symbolic memory is populated and exploited by a data science agent. In a first stage, heterogeneous machine learning artifacts are ingested, normalized, and integrated into a unified knowledge graph, providing a structured representation of datasets, tasks, implementations, and experimental results. This graph constitutes persistent, reusable analytical memory.

In a second stage, the knowledge graph is queried to retrieve task-relevant subgraphs, which are distilled into structured reports containing ranked candidate pipelines together with supporting evidence. Rather than exposing the full graph to the agent, these reports provide a compact, curated interface tailored to the current analytical task.

In the final stage, the report is consumed by an LLM-based agent as structured context. The agent uses this information to guide planning and code generation, optionally executing resulting workflows in a sandboxed environment. Execution results and evaluation metrics can be fed back into the knowledge graph, enabling the continuous accumulation of experience and closing the neuro-symbolic loop between reasoning, action, and memory.

Concrete Instantiation of Symbolic Long-Term Memory

MLSea [Dasoulas et al., 2024]: Symbolic Long-Term Memory. As argued in the previous section, effective data science agents require persistent, structured long-term memory to complement neural reasoning. In our implementation, this role is fulfilled by *MLSea*, which serves as the symbolic memory substrate for historical machine learning knowledge.

While Retrieval-Augmented Generation (RAG) can surface relevant textual context, data science tasks often require precise, relational knowledge capturing dependencies between datasets, tasks, implementations, hyperparameters, and outcomes. Such structure cannot be reliably inferred from unstructured text alone and motivates the use of knowledge graphs.

MLSea captures machine learning datasets, experiments, software artefacts, and scientific publications from *OpenML*, *Kaggle*, and *Papers with Code* into a unified knowledge graph. Its core components, the *MLSO* ontology, *MLST* taxonomies, and the *MLSea-KG*, provide a shared semantic vocabulary for describing machine learning processes, enabling explainable retrieval, comparison, and reuse of prior analytical experience.

AssistML AssistML [Zacarias et al., 2023] complements MLSea by addressing heterogeneity and incompleteness in the symbolic memory, providing a schema for comparison and recommendation across historical experiments. When exploring MLSea, we quickly observed a strong overall heterogeneity, i.e., varying quality of descriptions, incompleteness in different patches of the graph, heterogeneous notation, etc. Here, *AssistML* can provide valuable assistance. It proposes a metadata model that standardizes different datasets and task types, making them comparable and enabling it to suggest solutions for target metrics without relying on LLMs. Its original workflow is divided into four steps:

1. select ML solutions based on data similarity,
2. identify acceptable/nearly acceptable ML solutions,
3. find ML solution patterns and finally
4. generate a list of recommendations.

Integrated workflow

Ingestion Pipeline. We implement an ingestion pipeline, illustrated in Appendix D, to automate the extraction, transformation, and loading of *OpenML Datasets*, *Tasks*, *Flows*, and *Runs*[4] into a structured metadata repository. To reduce system complexity and ensure data quality, the current implementation is deliberately restricted to OpenML, which contains the majority of well-curated artefacts, whereas other sources exhibit higher degrees of incompleteness and heterogeneity.

Each dataset is first analyzed using a data profiler to capture structural characteristics such as feature types, dimensionality, and target attributes. Based on this analysis, the pipeline retrieves and processes associated ML tasks (e.g., classification, regression, clustering), thereby establishing explicit relationships between datasets and tasks. Subsequently, implementations

[4] https://docs.openml.org/concepts/, accessed on 02.05.2026

are processed by extracting metadata such as software dependencies, framework versions, and configuration parameters from OpenML. These implementations are stored in a document-oriented database and linked to the corresponding tasks.

For model-level processing, the pipeline retrieves experimental runs associated with each task and extracts performance metrics and hyperparameter configurations. Model setups are reconstructed by combining tasks, implementations, and hyperparameters into coherent experiment representations. Evaluation metrics derived from run data enable systematic performance comparison across models, datasets, and tasks. Together, these steps populate a structured, queryable representation of historical machine learning experiments.

Report Generation. Once the metadata repository is populated, the backend component shown in Appendix D generates structured analytical reports. Given a query defined by a dataset, a task, and optional user preferences, the system retrieves relevant historical experiments and produces ranked model recommendations.

The queried dataset is again processed by the data profiler to ensure consistent feature characterization. Based on dataset similarity, task compatibility, and user-defined optimization criteria (e.g., accuracy, runtime, or resource consumption), the recommender identifies suitable candidate models. Rather than exhaustively searching the configuration space as in traditional AutoML systems, the approach leverages accumulated historical metadata to rapidly narrow the solution space.

The resulting report provides a structured evaluation of candidate models, including rankings, performance metrics, and explanatory evidence derived from prior runs. Models are categorized into *acceptable* and *nearly acceptable* solutions, reflecting how closely they match the specified criteria. By explicitly exposing the rationale behind each recommendation, the system supports transparency and interpretability for users with varying levels of ML expertise.

Role in the Neuro-Symbolic Agent Architecture.

Together, MLSea and AssistML instantiate the *symbolic long-term memory* of the neuro-symbolic data science agent. They externalize analytical knowledge that would otherwise remain implicit in prompts or execution traces, enabling grounded perception, informed planning, and experience-aware decision-making.

In contrast to purely neural approaches, this design allows analytical knowledge to accumulate across tasks and users, forming a stable substrate for reproducible and explainable automation. The evaluation in chapter 8 examines how this symbolic augmentation compares to the previously introduced AutoMLWrapper, quantifying the impact of structured long-term memory on data science workflows.

6.3 Conclusion

This chapter addressed RQ4 by demonstrating how SEDAR can empower non-technical users to perform complex data science workflows through a combination of automation, semantic structure, and agent-based interaction. Rather than treating automated machine learning as an isolated capability, the chapter positioned analytics as an integrated, end-to-end process spanning data access, experiment execution, result interpretation, and knowledge reuse.

The *AutoMLWrapper* provides a practical abstraction layer over heterogeneous AutoML frameworks, allowing domain experts to execute reproducible model training workflows with minimal manual configuration while preserving traceability and comparability. Building on this foundation, the chapter introduced *Data Science Agents* as a higher-level interaction paradigm. By decomposing agent behavior into perception, reasoning, planning, and action, and by explicitly separating decision-making from execution, these agents reduce the cognitive and technical burden traditionally associated with data science tooling.

A central contribution of this chapter is the formulation of a *neuro-symbolic architecture* for data science agents. By distinguishing between short-term, task-specific context and persistent symbolic long-term memory, the architecture clarifies how LLM-based agents can be grounded, constrained, and made reproducible. The implementation of a KG-enhanced agent demonstrates how structured historical knowledge can guide planning and model selection beyond transient prompt-based reasoning.

Taken together, the presented components enable non-technical users to derive value from heterogeneous industrial data while operating within a controlled, transparent, and extensible analytics environment. Rather than replacing human expertise, SEDAR augments it by externalizing analytical knowledge, standardizing workflows, and enabling experience-aware automation. This establishes a foundation for scalable, explainable, and increasingly autonomous data science in chemistry-driven industrial settings.

The various presented modules are evaluated in detail on use case from our research domain situated in data-driven chemistry in section 8.5.

Dimension	Search Engines	Knowledge Graphs	Large Language Models
Correctness	+ Accurate results - Prone to noisy content	+ Accurate answers + Precise operators	- hallucinations
Coverage	+ Broad data coverage	+ Domain-specific data - Limited to curated data	+ Broad data coverage - May lack depth in niche areas
Completeness	+ Complete search results - May miss relevant context	+ Complete, relevant answers - May miss relevant context	- Often incomplete - May miss relevant context
Generation	- No content generation	+ Can infer new knowledge by ontologies/rules & graph-learning	+ text generation
Synthesis	- Cannot combine multiple sources - Isolated results	+ Integrates data from multiple sources + Synthesizes structured data	+ Combines diverse sources into answers + Produces integrated outputs
Transparency	+ Results traceable to data - Some rankings may be opaque	+ Algorithmically explainable + Results explainable via graph data	- Opaque decision-making - No insight into reasoning
Refinability	+ Refinable by query adjustments - Complex rank refinement	+ Refinable by graph modifications	- Refining responses is indirect - Unpredictable quality improvements
Usability	+ Natural language search + Supports simple queries	- Requires structured queries - Difficult for complex queries	+ Easy conversational input + Intuitive for non-experts
Efficiency	+ Fast retrieval for simple queries + Quick for basic queries	+ Efficient for structured queries - Slower for complex tasks	- Computationally expensive - Heavy resource usage

Table 6.1: Comparison of Search Engines, Knowledge Graphs, and LLMs across different dimensions relevant for automating data science, adopted from Hogan et al. [2025].

7

A Multi-Agent Natural Language Interface for SEDAR

"Simplicity is the ultimate sophistication."

Leonardo Da Vinci

This chapter addresses the final component of this dissertation's overarching objective: to enable automated and intelligible data management in heterogeneous, semantic data lake environments. While the previous chapters established SEDAR's architecture, semantic data management capabilities, and automated analytics pipeline, the system still required users to engage with technical interfaces, APIs, and workflows. Such interaction models presuppose substantial technical expertise and therefore limit the participation of domain experts who lack detailed knowledge of data engineering, query languages, or system orchestration.

The central motivation of this chapter is therefore to investigate how LLM-based agents can be used to provide an intuitive natural language interface (NLI) that executes complex data management and analytics tasks on behalf of the user. In doing so, this chapter directly addresses RQ4 by examining how non-technical users can be empowered to perform complex data management and analytics tasks through natural language, without requiring explicit knowledge of underlying data models, query languages, or execution frameworks.
This chapter is based on [Hoseini et al., 2026a].

Chapter Outline

This chapter is structured as follows:

- Section 7.1 - Motivation for the NLI
- Section 7.2 – Background and related work on LLM-based agents, tool calling, RAG, and NLIs for data systems.
- Section 7.3 — Extension of SEDAR architecture and motivation for an NLI that provides unified access to all system components.
- Section 7.4 – Conceptual foundation and architecture of the multi-agent system.
- Section 7.5 – Realization of the system, including agent workflows, retrieval pipeline, chat interface, and finetuning.
- Section 7.6 – Quantitative and qualitative evaluation of the NLI, including correctness, robustness, and reasoning metrics.
- Section 7.7 – Concludes the chapter by assessing how the NLI contributes to answering the research questions (section 1.2).

7.1 Introduction

Despite offering powerful capabilities, data lakes often remain inaccessible to domain experts and non-technical users due to steep learning curves, limited interfaces, and complex workflows. This usability gap prevents organizations from broad adoption and from fully leveraging their data assets [Ravi and Ayyagari, 2023, Sawadogo and Darmont, 2021, Bianchini et al., 2024] and motivates the need for more intuitive, self-service interaction paradigms [Hechler et al., 2023].

Recent developments in natural language processing and the emergence of LLMs have opened new avenues for simplifying data interactions [Brown et al., 2020]. Various works [Patil et al., 2024, Xi et al., 2025, Wang et al., 2024, Hollmann et al., 2024, Hong et al., 2024] have demonstrated the viability of equipping LLMs with agentic behaviors to perform data-intensive tasks, from question answering to data transformation and ML pipeline orchestration.

In this chapter, we explore how such agents can be leveraged to create a more intelligent interface for semantic data lakes that relieves users from the burden of navigating intricate dialogues to configure sophisticated workflows. The proposed natural language interface (NLI) enables users to perform complex data management tasks using everyday language. The interface facilitates general data management tasks (such as data ingestion, discovery, integration, processing, and ML) by leveraging the ability of LLMs to convert user queries into actions executed on the system. This work introduces a novel multi-agent, LLM-driven interface that explicitly targets natural language interaction, data integration, and exploration in semantic data lakes. Unlike prior systems that hard-code query templates or narrow use cases [Liu and Xu, 2025], our NLI introduces a transferable design pattern: decomposing queries into modular agent tasks, grounding them in API calls of the underlying system, and evaluating them with reasoning-focused metrics.

While approach is applied in SEDAR, the main contribution is the methodology itself: decomposing natural language queries into structured function calls over large APIs. The same pattern could be applied to other domains, such as the Azure SDK, where hundreds of client libraries and operations present a similar challenge for non-expert users. In summary, our contributions are twofold:

1. **Methodological contribution:** We introduce a transferable multi-agent design (Manager–Tool/Code–Synthesizer) applicable to any API-centric platform, together with an evaluation protocol including (i) semantically varied query datasets, (ii) parameter-efficient fine-tuning for adaptation, and (iii) metrics for reasoning consistency and robustness.
2. **System contribution:** To verify the proposed multi-agent architecture, we have implemented an NLI that integrates LLMs with retrieval and tool execution for semantic data lakes that enables end-to-end tasks such as ingestion, discovery, AutoML and semantic data management.

Datasets and code to reproduce the results can be found on Github.

7.2 Related Work

LLM for data management have been shown in research to be effective for key tasks such as data discovery and related dataset search, semantic labeling, and automated metadata generation [Li et al., 2024]. Leventidis et al. [2024] evaluate a large-scale dataset for semantic table search comparing different methods such as Starmie [Fan et al., 2022], which proposes representation learning for data discovery using LLMs. NumSearchLLM [Subramaniam et al., 2023] combines LLMs with knowledge graphs to improve semantic labeling and related table search for purely numeric tables in enterprise data lakes. CHORUS [Kayali et al., 2023] uses LLMs for data discovery tasks demonstrating superior performance over task-specific models and human experts. In [Peeters and Bizer, 2023] it is shown that ChatGPT can effectively perform entity matching, suggesting that LLMs can be used for semantic labeling. TabMeta [Tan et al., 2025] uses LLMs to generate comprehensive metadata for tabular data, for data discovery and search to improve the performance of other downstream tasks such as machine learning pipelines [Giner-Miguelez et al., 2024]. LEDD [An et al., 2025] facilitates data discovery in data lakes by building hierarchical global catalogs and supports semantic table search, improving understanding and organization of large-scale datasets. Evaporate [Arora et al., 2023] enables structured data extraction in data lakes via code synthesis. Gomez [Gómez and Fillottrani, 2024] explore the potential of LLMs to perform OBDA automatically.

LLM Tool-Calling refers to LLMs that invoke external tools, APIs or functions such as a calculator or database query engine, or to request external information that might otherwise not be available in the training data [Yang et al., 2023]. Gorilla [Patil et al., 2024] and Toolformer [Schick et al., 2023] are approaches where finetuning was used to create a model specialized in API tool calling, while He [2024] show that LLMs can be taught to call external tools without any finetuning. The Model Context Protocol [Hou et al., 2025] is a standardized interface introduced by Anthropic designed to enable seamless interaction between AI models and external tools

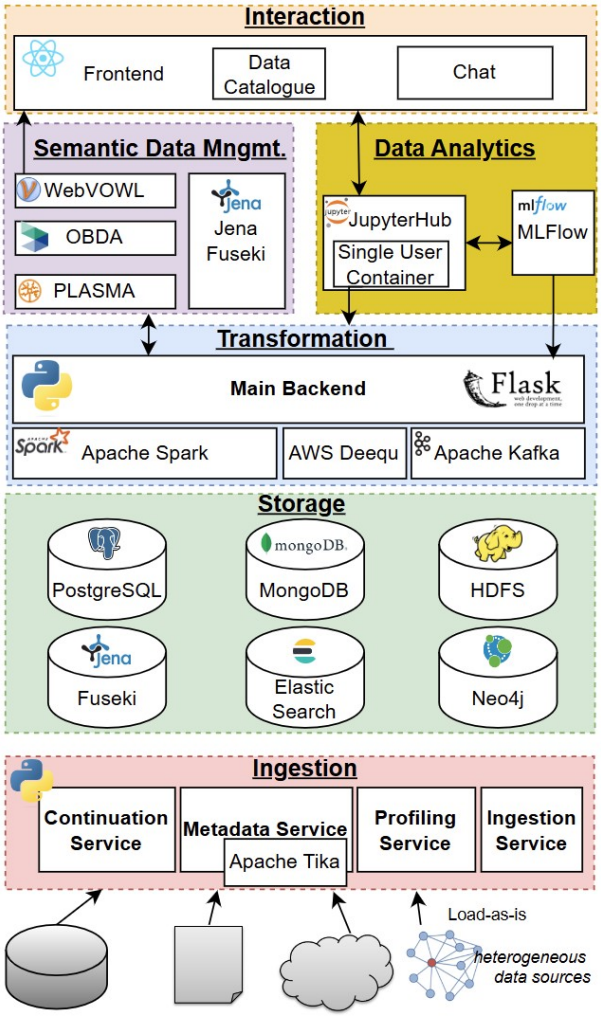


Figure 7.1: SEDAR's architecture

1. *Interaction*, to provide tools enabling users to interact with the data.

2. *Data analytics*, to generate and manage machine learning models crafted from the stored data.

3. *Semantic data management*, to enrich data sources semantically for ontology-based data access.

4. *Transformation*, to provide a scalable environment to preprocess, integrate, and transform data.

5. *Storage*, to efficiently store the data in big data systems as a polystore [Duggan et al., 2015].

6. *Ingestion*, to continuously extract data from heterogeneous sources (e.g., files, databases, streams).

and resources.

Natural Language Interfaces for Databases [Koutrika, 2024, Liu and Xu, 2025] allow querying databases in natural language instead of technical query languages like SQL [Li et al., 2023a]. LOTUS [Patel et al., 2024] adds semantic operators to the relational model, enabling complex queries over structured and unstructured data with language models. SYMPHONY [Chen et al., 2023b], on the other hand, enables natural language querying of multi-modal data lakes without prior integration. QueryArtisan [Tang et al., 2024] enable explainable, multi-modal data exploration in natural language for data lakes. DataScout [Chen et al., 2025] introduces an interactive system where an LLM-based agent acts as an NLI for data platforms.

7.3 Extending the SEDAR Architecture

In chapter 4 we introduced SEDAR's original four-layer architecture, consisting of ingestion, storage, interaction, and analytics. The subsequent chapters on semantic data management (chapter 5) and automated machine learning (chapter 6) have shown how has SEDAR has

evolved beyond these initial components, which now includes two additional conceptual layers: a *semantic data management layer* and an *intelligent analytics layer*. This architecture, indicating the various technologies used, is illustrated in Figure 7.1, which is divided into six functional areas.

SEDAR is deployed in Industry 4.0 settings in the chemical domain [Hoseini et al., 2024e,c]. This requires an external library that abstracts the backend API to simplify integration with industrial processes, allowing developers to focus on building automation solutions rather than managing low-level API details. This programmatic interface for the backend is called the *SEDAR-API* (see section 4.3.4).

Both the SEDAR-API and SEDAR's frontend remain a critical bottleneck for many users. Interacting with these interfaces often requires familiarity with internal data structures, functions, APIs, and workflows, posing a significant barrier not only for non-technical users but also for analysts and researchers. To truly democratize access to the system, there is a need for a more intuitive interaction. An NLI bridges this gap by allowing users to express their information need in everyday language. By converting natural language from a chat into actionable commands, the NLI lowers the cognitive and technical threshold required to engage with the system.

To cover SEDAR's functionality as much as possible we define a set of representative queries (see figure 7.2). For example, the first query in *Data Discovery* requires first retrieving all available datasets, and their metadata to then reason for relatedness on the gathered information. Each query maps to backend API functions. For machine learning tasks, this includes dataset uploads (with labels for classification/regression on tabular and image data) and the creation of AutoML models via the *AutoMLWrapper* module (see section 6.1.2). The last items in Semantic Data Management are the running example for section 7.5.3. To better approximate real-world use, we additionally introduce variations (see appendix E.1) of the representative queries in the evaluation to test robustness against linguistic variations.

7.4 Multi-Agent System: Concepts and Architecture

A central challenge in designing multi-agent systems is balancing autonomy and alignment: agents must produce outputs consistent with system goals while retaining sufficient flexibility to handle diverse tasks in dynamic environments [Händler, 2023].

In our setting, an agent is an LLM-based component that reasons over the system state and interacts with the SEDAR API to perform task-specific actions. The overall system is organized as a directed graph (see Figure 7.3), where each node corresponds to an agent invocation. Agents operate with dedicated prompts, access shared state and tools, and communicate indirectly as defined by the workflow.

A key design decision concerns the trade-off between retrieval-based approaches and explicit task decomposition across multiple agents. Pure retrieval simplifies orchestration but limits flexibility, while highly specialized agents increase coordination overhead.

To address this, we adopt a hybrid architecture that combines dynamic retrieval with lightweight coordination. API classes are registered via decorators, and their public methods, implemen-

Ingestion and Data Discovery:
- Ingest my file into the appropriate storage system.
- Upload the datasets X & Y and run their ingestion concurrently.
- Retrieve the versions of dataset X? What are their differences?
- What is the lineage of dataset X?
- Look at the dataset X and add a description to it.
- Given the dataset X, find all datasets related to it.
- Given the dataset X, find all datasets with columns that we can potentially join with the columns of dataset X.
- Name all datasets that contain data about chemicals.

Machine Learning:
- Use the dataset X and create an AutoML run to predict the target T.
- Perform binary classification based on dataset X to predict T.
- Which ML experiments exist for the dataset X? Explain them.

Semantic Data Management:
- Create a new ontology O by uploading the file F.
- Find a fitting semantic label from Ontology O for dataset X.
- Create a semantic model to join the datasets X, Y and Z.
- Using the OBDA functionality join the datasets X, Y and Z.

Figure 7.2: Representative queries for ingestion, ML, and semantic data management.

tations, and doc strings are embedded into a vector store. This enables class-based retrieval of relevant functionality at runtime Lumer et al. [2024], avoiding excessive agent fragmentation while maintaining scalability: new capabilities require only embedding updates rather than architectural changes.

The architecture comprises six agents (see Table 7.1) and is organized into two layers: The outer loop is executed only once and handles query decomposition and the final response, while the inner processes subtasks through a Manager that conditionally invokes Code/Tool Agents, with outputs integrated by the Synthesizer.

This particular agent decomposition was guided by the observation that retrieved tool and knowledge subsets tend to cluster around distinct functional and semantically coherent domains such as ingestion, dataset handling, or AutoML.

All agents use RAG to access the context about the API relevant to the task. Tool Agents operate over a narrow, class-specific scope, reducing the number of functions the LLM must reason about at once. Execution feedback, including intermediate outputs, results and errors, is passed among agents to refine outputs iteratively, following the ReAct pattern [Yao et al., 2023]. This dynamic design removes duplication and eliminates the complexity by limiting the number of tools or functions the LLM must reason about at once, simplifying tool selection and reducing the cognitive load on the system. All public API functions are embedded into a vector store, enabling dynamic, on-demand retrieval of relevant API operations. The Tool Agent, in contrast, is tightly scoped to a specific API class instance. It operates on a much narrower API surface, but it still performs retrieval over the methods of that class. Although some agents (e.g., Manager) still require a global API view, the overall architecture is simpler,

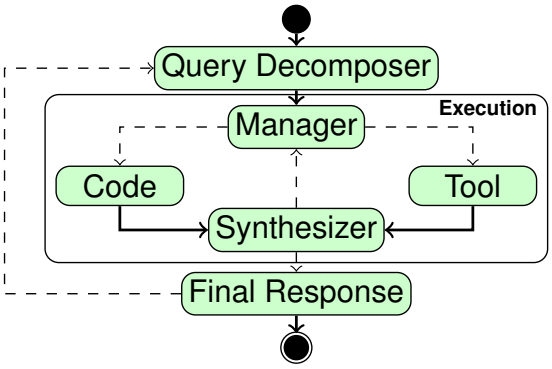


Figure 7.3: Workflow of the multi-agent system. Solid arrows denote core task flow, dashed conditional paths.

Agent	Role	Input	Output
Query Decomposer	Break query into subtasks	User query, API context	Subtasks
Manager	Assign subtasks	Subtasks, cache	Tool / code decision
Tool	Execute API calls	API context, cache	API result
Code	Run custom code	API context, cache	Code result
Synthesizer	Merge results	Agent outputs	Final response
Final Response	Deliver response	Synthesizer summary	User reply

Table 7.1: Agent overview.

more maintainable, and better equipped to scale. The workflow aims to handle complex queries reliably and efficiently through a combination of systematic decomposition, retrieval, specialized agent collaboration, and robust error handling.

7.5 Realization

We describe the multi-agent system, retrieval pipeline, chat interface, and conclude with fine-tuning & prompt compression.

7.5.1 Multi-Agent System

Every agent has some *state* that is passed between the nodes or agents used to maintain context and facilitate communication.

```
class MainState(BaseState):
    query_decompose_agent_messages: Annotated[list]
    final_response: str
    decomposed_queries: list[str]
    execution_agent_state: ExecutionAgentState
    has_reached_recursion_limit: bool
```

Listing 7.1: The main state.

To give an example, for the overall main (figure 7.3) we use Listing 7.1. The state is mostly used to store outputs from the agents, such as the decomposed queries, and the final response. The decomposed queries are passed into the `Executionstate` (Listing 7.2) of the Execution loop, which can then be accessed globally. Here, we specify a recursion limits for the number of iterations the Execution loop can run to prevent infinite loops. For example, consider the query: “Update the title of the ’Physics Experiments’ dataset to ‘Physics Dataset’.” Assuming the dataset has already been found (e.g., via a search function), the cache may contain:

```
{ "_WORKSPACE_659dee04": "Workspace(id='...', title='...')",
  "_DATASET_9c4014d3": "Dataset(id='...', title='...')",
  "_LAST_OUTPUT": "..." }
```

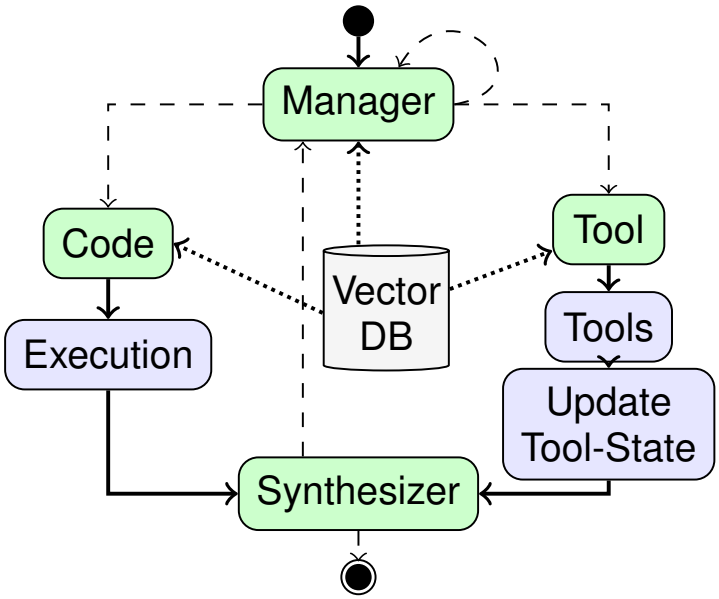


Figure 7.4: Execution loop. Green=agents, blue=Python functions. Dashed/dotted edges indicate conditional flow/RAG.

```python
class ExecutionState(BaseState):
    current_query: str
    next_action: str
    code_agent_messages: Annotated[list, add_messages]
    manager_agent_messages: Annotated[list, add_messages]
    tool_agent_messages: Annotated[list, add_messages]
    tool_execution_messages: Annotated[list, add_messages]
    code_execution_messages: Annotated[list, add_messages]
    synthesize_agent_messages: Annotated[list, add_messages]
    object_cache: dict[str, Any]
    decomposed_queries: list[str]
```

Listing 7.2: The state of the Execution loop.

Cache keys follow the pattern: underscores + class name (uppercase) + short unique identifier. In this case, the `Dataset` object was stored under both `_LAST_OUTPUT` and `_DATASET_9c4014d3`. Since a relevant update function likely exists for the `Dataset` class, the Manager Agent selects the Tool Agent and includes the object reference:

```
{ "action": "TOOL",  "tool_object": "_DATASET_9c4014d3"}
```

Alternatively, the Code Agent would also be a valid choice.

Every incoming user query first reaches the *Query Decompose Agent*. This agent decomposes the user query into atomic subtasks based on information about the API retrieved from a vector store. In the best case, each subtask can be answered by a single API call. For this agent, we use a mix of top-7 retrieval and passing only the names of the methods to the LLM, so that the agent then knows about relevant classes of the API and their methods. For example, for the query“Show me all datasets”, the `get_all_datasets` function from the `Dataset` class should be in the top-7 retrieved functions. These methods and their classes are passed to the LLM together with the descriptions and information about the parameters. All other API classes

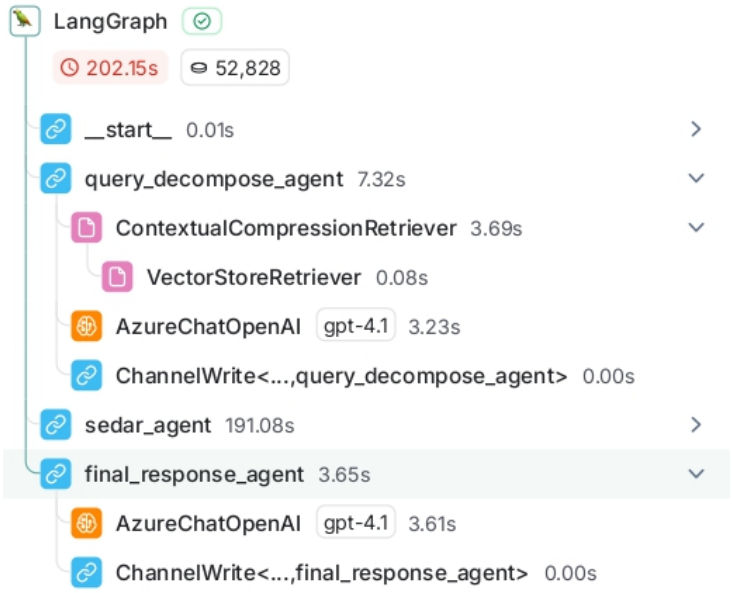


Figure 7.5: LangSmith trace for *"Get all datasets"*

are also listed with descriptions and for all their methods, we only pass the method header (name and parameters). This way, the LLM still knows about all methods but the number of tokens in the prompt is limited emphasizing the most important functions only.

Figure 7.4 depicts the *Execution loop* that is invoked inside the main loop in step four of the workflow. The next unanswered subquery is retrieved from the state and passed to the *Manager Agent*. The retrieval pipeline provides relevant context by identifying the top seven candidate functions, alongside class information and the names of all available API functions.

Executed functions or code often return instances of API classes, which play a central role by storing intermediate objects from function calls or code executions in an `object_cache`. These cached instances can later be reused by the Tool or Code Agents as parameters or inputs in subsequent steps. In addition to the object cache, various message arrays serve as memory structures that retain the interaction history of each agent to inform and guide decisions. However, not all messages are relevant to every agent at every step. For example, the Code Agent may only receive messages from the Manager, while the Synthesizer aggregates messages from all agents to form a final response. This selective context passing is made possible through the LangGraph framework, which allows fine-grained control over the shared state.

The Manager determines the next action by analyzing the current query, the retrieved function context, and the available objects in the cache. As depicted in Figure 7.4, the Manager has three possible outgoing conditional edges: The Tool Agent is preferred when a matching function is available, as it is more reliable and operates on a specific object from the cache, while the Code Agent is more flexible and capable of arbitrary code generation, but is also more error-prone. If the Manager produces an invalid output, such as failing to select either TOOL or CODE, generating invalid JSON, or referencing an object that is not present in the

cache, the self-loop edge is taken. This conditional routing enables the Manager to robustly handle errors and adaptively select the most appropriate step based on the current context and state. In appendix E we provide the implementation details for the Manager, including its system prompt, prompt template, and few-shot examples that define how it coordinates workflows and selects the next best action.

The *Tool Agent* receives the current query, the object cache, and the API context selected by the Manager. Based on this, the Tool Agent determines which function to invoke on the selected objects. The system avoids listing functions in the prompt by using native tool-calling features of model providers (e.g., Ollama, OpenAI, Anthropic), where functions are registered in advance. Function calls may reference cached objects. After execution, the agent updates the global state—logging messages and the cache. If the function fails or the query remains unresolved, prior attempts and outputs are utilized for iterative refinement. Due to numerous parameters or intricate behavior, some API functions are unsuitable for direct use by the Tool Agent. To address this, *custom tool agents* are dedicated agents that ultimately invoke an equivalent of the original function. One example is shown in section 7.5.3.

The Code Agent is selected by the Manager when a query cannot be resolved by a single API call or requires complex logic such as attribute access, conditionals, or composing multiple APIs. When invoked, it receives the same context as the Manager, i.e., the seven most relevant functions from the vector store, information about the associated API class, and the names of all available functions to generate custom code. The Code Agent also has access to the full object cache. The agent is equipped with helper functions to load, store and update objects, typically via a custom `print` function that preserves intermediate results. The generated Python code is executed in the Code Execution node, after which the system state is updated by logging the output and storing resulting objects in the cache. If the code fails or doesn't resolve the query, the previous code and its output are included in the next iteration.

The Synthesize Agent is the final component in the Execution loop. Its role is to determine whether the current query has been successfully addressed and to generate a user-friendly summary. It receives the full context: the initial and current queries, decisions by the Manager, steps taken, and resulting outputs. If the query is fulfilled, the Synthesizer outputs a summary and routes control to the end node. Otherwise, if errors occur, the result is insufficient, or an incorrect function was used, it passes control back to the Manager to retry using the updated context and feedback. By acting as a verification and approval stage for each subtask, the Synthesizer offloads responsibility from the Manager.

7.5.2 Retrieval Pipeline

In our system, agents are entirely dependent on the candidates of the retrieved functions to construct meaningful responses. If a relevant function is not retrieved, the agent will either fail to answer or hallucinate functionality. Hence, a strong retrieval pipeline is essential to ensure high-quality responses.

The retrieval pipeline in our system consists of four main parts as shown in figure 7.6: the

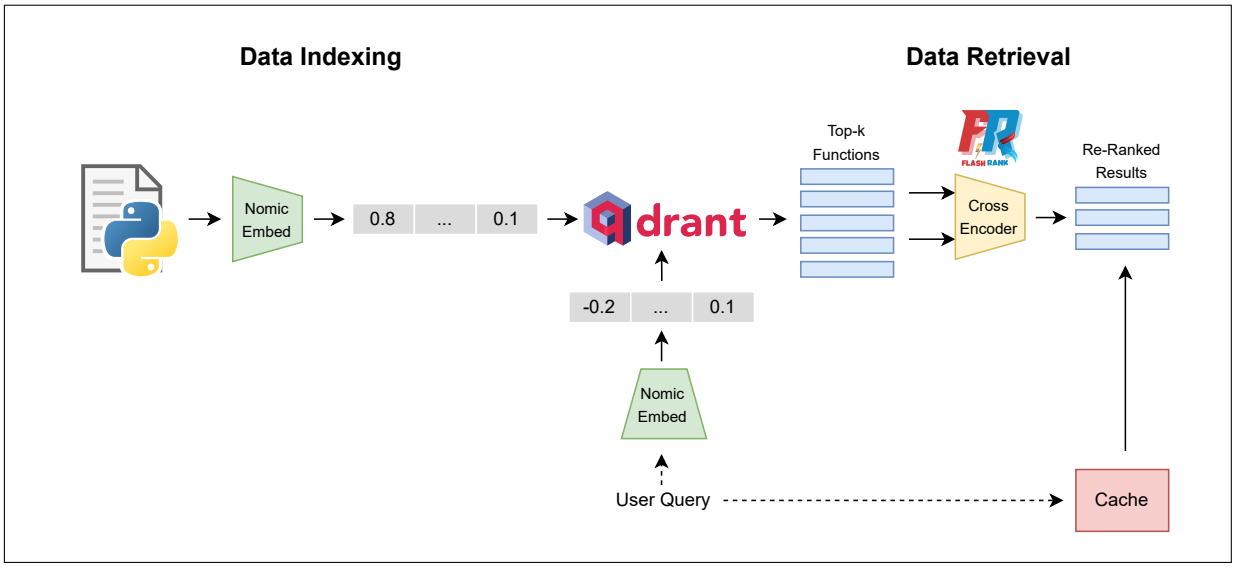


Figure 7.6: The full retrieval pipeline. The left side shows the indexing of the API functions and the right side the retrieval process including the re-ranking and caching.

indexing of the API functions, the vector store used for storing and retrieval, the cross-encoder for re-ranking, and the cache. Before any retrieval for a given query is performed, we check if the retrieval results are already in *cache*. If a cache hit occurs, the stored results are returned immediately, avoiding unnecessary computation and thereby improving response time.

Then the API functions need to be *indexed*. For this, we use a custom Python decorator (`@cacheable`) that registers the classes to be indexed. All public methods of each registered class that are not excluded explicitly are then indexed. For each function, a document is created that contains the full function source code, i.e., the function signature, the doc string, and the function body. It also includes metadata such as the class name, the function name, and an ID. For *embedding* we rely on *Nomic Embed* [Nussbaum et al., 2025]. With a context length of 8192 tokens, it outperforms OpenAI's Ada-002 and text-embedding-3-small on both short and long context tasks and ranks well on the MTEB [Muennighoff et al., 2023] leaderboard for retrieval tasks. In our initial tests, it has shown better performance than other models such as BGE M3 [Chen et al., 2024] or OpenAI's text-embedding-3-large. It has a large enough context length to embed any of our API functions and can be hosted on our infrastructure for which we use a Qdrant vector store [GmbH, 2024]. The dimension of the embedding vectors is 768.

The main *retrieval* process uses cosine similarity for similarity search. The retriever is employed in two main ways within our system. First, general retrieval of functions from the API is used by both the Query Decompose Agent and the Manager. This method allows specifying the search query, the parameter k (denoting the number of top-matching documents to retrieve), and whether to include short descriptions of other relevant methods and classes not directly retrieved. Second, tool retrieval for a specific class is provided to the Tool Agent. In this case, a class and its associated tools are provided, and only those tools that match the functions retrieved for the given query with the parameter returned. In the initial retrieval to avoid missing any relevant functions, a larger k value such as 20 worked well for us but can be tuned based

on system size or domain specificity. To enhance retrieval precision, we apply re-ranking using FlashRank [Damodaran, 2023] on retrieved functions before passing them to the LLM to narrow down the top-7 results.

Figure 7.5 shows a publicly available LangSmith trace for the *"Get all datasets"* query, recording the agent workflow including prompts, retrieval results and outputs of *Cohere's Command-A LLM*[1]. The *ContextualCompressionRetriever* component reduces the number of candidate documents (from 20 to 7) to filter out less useful information, surfaces the most pertinent content and reduce context length for downstream tasks: In this trace the retrieval correctly ranks the `get_all_datasets` function first, which is later on registered for and called by the tool agent. Each function is retrieved with its doc strings which documentation on in- & output variables and example usage.

7.5.3 Chat Interface & Example Walkthrough

The interface for the user to interact with the system conversationally is implemented via Chainlit [Literal AI, 2024] which seamlessly integrates with LangGraph. Figure 7.7 shows the new chat interface integrated in the existing SEDAR frontend. The chat window shows an intermediate step from the example below, where the system suggests some labels for datasets that can be accepted or declined through the UI. The action bar allows files to be uploaded, for example, to create a new ontology/dataset. Different settings can be changed such as temperature, model provider and models and whether to use reasoning effort with reasoning models [OpenAI, 2024b, Guo et al., 2025].

The chat interface consists of a frontend and a backend. The frontend of the chat interface is integrated as a whole into the existing frontend to communicate the authentication or to pass other information such as the current workspace and more. The Chainlit backend wraps the LangGraph multi-agent system and communicates with the frontend only. Communication with the SEDAR's backend only happens from within the multi-agent system through the SEDAR-API.

To illustrate the multi-agent workflow in practice, we present a walkthrough of a representative user query and the corresponding system steps. The example query resembles the use case shown at CIKM'23 [Hoseini et al., 2023]:
Create a semantic labeling for the Countries, Capitals and Currencies datasets. Use the DBpedia ontology for labeling. Convert it into a mapping and perform OBDA to find which capitals use which currencies.
This query requires the system to search for specific datasets, locate an ontology, create a semantic model, perform attribute labeling, convert the semantic model into an RML mapping [Dimou et al., 2014], and create an OBDA query that joins the data involving several interdependent subtasks:

[1] `https://cohere.com/blog/command-a`, accessed on 02.05.2026

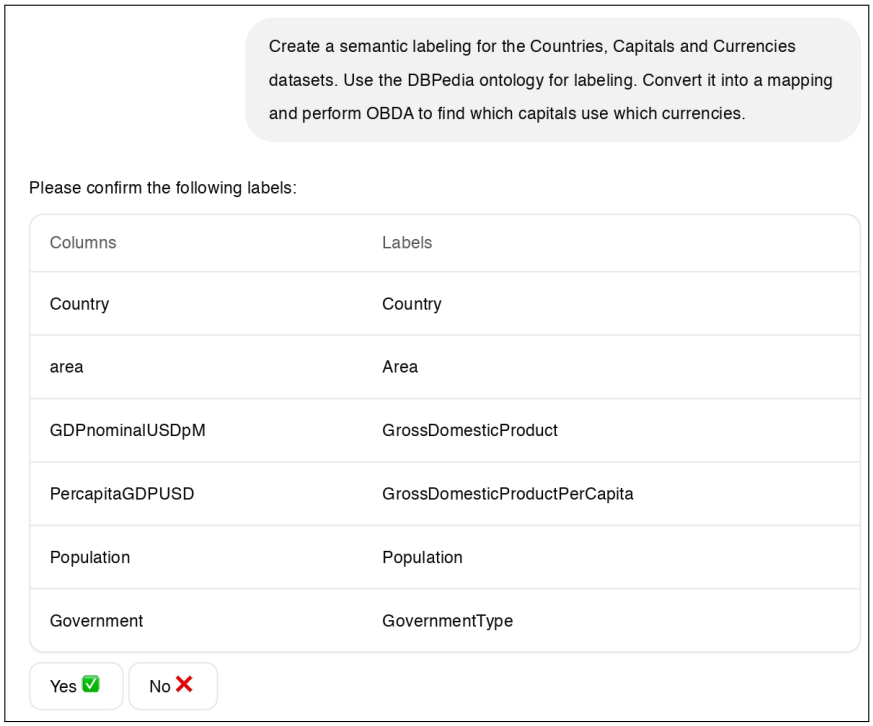


Figure 7.7: The new chat interface integrated into the frontend. The first step involves user confirmation for the predicted semantic labels of a dataset.

Step 1 - Query Submission: The user submits the query through the chat interface to initiate processing.

Step 2 - API Function Retrieval: The system retrieves relevant API functions from the vector store using the retrieval pipeline (section 7.5.2), and re-ranks to prioritize the most relevant candidates such as `label_modeling_attributes`, `datasets_search`.

Step 3 - Query Decomposition: Decomposing the query via the retrieved context into atomic subtasks stored in the system state and passed to the next agents:

- Search for the dataset *Countries* / *Capitals* / *Currencies*
- Get all ontologies and find the DBpedia ontology.
- Create a semantic model for the datasets.
- Label the attributes of the datasets using the DBpedia ontology.
- Convert the semantic model into an RML mapping.
- Create and execute an OBDA query that joins the data and returns the capitals and their currencies.

Step 4 - Subtask Execution: Each subtask is processed through the Execution agent loop. API functions contextually relevant to the current subtask are retrieved and re-ranked. The Manager determines whether the subtask can be solved by calling an API function or requires generating custom code. In this example, the Tool Agent is always chosen, as all the subtasks can be solved by single API functions. Some functions, such as `execute_obda_query`, are implemented as custom tool agents to handle complex tasks that require multiple LLM invocations or function calls. For the final subtask, the Tool Agent invokes this custom agents, which

orchestrates the OBDA query generation and execution process. The Synthesizer evaluates whether the subtask was completed. If not, the agents can iteratively refine their approach or fix errors.

Step 5 - Response: Once all subtasks are completed and validated, the Final Response Agent aggregates the results and generates a user-friendly summary. The example's response is:
I've completed the semantic annotation (using the DBpedia ontology), generated an RML mapping and ran an OBDA SPARQL query to link capitals to their currencies. Here are the first ten as examples:
1. Albania: Capital - Tirana, Currency - Lek . . .

This example demonstrates how the multi-agent system systematically decomposes a complex query, coordinates specialized agents, and maintains state to deliver accurate and user-aligned results, even for tasks requiring multiple steps and custom logic.

7.5.4 Finetuning & Prompt Compression

In our experience, even models with $>$70B parameters often struggle with complex and long-context tasks like tool calling, structured data handling, code generation, and multi-agent coordination which require specialized reasoning that can't be reliably performed out-of-the-box. By finetuning state-of-the-art LLMs on a custom dataset that closely matches these workflows, we aim to improve the handling of extended contexts and specialized tasks.

With SEDAR reset and populated by sample data, we run the multi-agent system against representative queries (figure 7.2). For these runs, we use high-performing "teacher" models (GPT-4o & Gemini2.0 [OpenAI, 2024a, DeepMind, 2025]) and fetch successful LLM calls. Each run contains a complete LLM interaction including the system message, any tool definitions, the prompt as well as any context about the object cache or API functions, and past messages from other agents. First, the data is stored in the more generic and then converted into a model-specific format. All entries are then manually reviewed to ensure high quality resulting in a dataset of 535 samples reflecting the types of reasoning tasks performed by agents in our system, i.e., tool calling, code generation, and response synthesis.

We finetuned two models with an Intel Xeon 8468 Sapphire CPU, 450 GB RAM, and a single NVIDIA H100 GPU with 96 GB VRAM. We employed QLoRA [Dettmers et al., 2023] for parameter-efficient finetuning. Configuration details can be seen in Table 7.2. Maximum sequence length was set to 65,536 tokens and the batch size per device was set to 2, with gradient accumulation yielding effectively 16.

The training strategy used selective masking, i.e., each training sample included the entire conversational history, but only the final assistant response was unmasked for loss computation to ensure models learned to interpret long, multi-turn contexts to generate the correct output. Early experiments without adequate masking or with shorter sequence lengths (e.g., 2,000 tokens) led to hallucinations and performance degradation, highlighting the importance of these design choices. Upon completion of training, we exported the LoRA adapter weights and merged them into the base models using 4-bit quantization. This allowed local inference

Configuration	Llama 3.3 70B	Qwen 2.5 72B
LoRA Rank (r)	32	64
Epochs	2	3
Trainable Parameters	414M (1.13%)	842M (2.19%)
Adapter Weight Size	1.5 GB	3.1 GB
Training Time	104 min	156 min
Quantized Size	$\sim$40 GB	$\sim$44 GB

Table 7.2: Finetuning configurations and results

on a single NVIDIA Quadro RTX 8000 GPU (48 GB VRAM).

To address the challenges of long prompts in our multi-agent system, such as the *lost in the middle* effect [Liu et al., 2024b], we incorporated *prompt compression* [Jiang et al., 2023]. Especially with smaller models, important context can be overlooked when buried in the middle of long inputs. Prompt compression reduces the number of tokens while aiming to retain critical information. LLMLingua [Jiang et al., 2023] computes token importance using a smaller LLM and removes low-importance tokens, producing compressed inputs that remain interpretable for the model but are often unintelligible to humans. To preserve reliability, we apply compression selectively, with a moderate compression rate (retain 60%) to balance efficiency and accuracy. User queries, retrieved API responses, function signatures, and cache object keys are excluded from compression to avoid errors. Only compressible content such as doc strings can be reduced in size.

7.6 Evaluation & Discussion

We conduct a quantitative and qualitative evaluation and discuss the results, including the impact of retrieval quality.

7.6.1 Quantitative evaluation

To rigorously assess the performance and robustness of our multi-agent LLM system, we conduct a **quantitative** evaluation based on a set of natural language queries. These queries cover the system's key functionalities and are semantically varied to test generalization beyond fixed templates. A key goal of this evaluation is not only to check whether the final answer is correct but also how robust the system is to linguistic variation-specifically, whether it can correctly interpret semantically equivalent queries phrased in different ways. This reflects real-world usage, where users may express the same intent using varied phrasing. The guiding question here is: **How consistent is the performance when users phrase the same request differently?**

We began by manually deriving 27 base queries that span major functional areas from the representative queries (see figure 7.2). For each base query, four semantically distinct

Model	**TOK** ↓	**ACC** ↑	**LACC** ↑	**P@1** ↑	**P@2** ↑
llama3.3:latest	25,455	0.12	0.16	0.12	0.26
finetunedllama:latest	**24,093**	0.16	0.16	0.21	0.31
qwen2.5:72b	32,959	0.30	0.35	0.27	0.40
finetuned-qwen:latest	34,959	0.33	0.31	0.28	0.43
command-a	42,015	0.87	0.84	**0.93**	**0.96**
gpt-4o-mini	35,428	0.70	0.70	0.77	0.82
gpt-4.1-mini	29,896	0.85	**0.87**	**0.93**	0.95
gemini-2.0-flash	31,656	0.67	0.68	0.73	0.74
o4-mini-high	50,466	**0.90**	0.84	0.91	0.95
gpt-4o	33,922	0.81	0.76	0.83	0.85
gpt-4.1	32,709	0.87	**0.87**	0.91	0.92
claude-3-7-sonnet	42,321	0.89	**0.87**	0.92	0.93
gemini-2.5-pro-preview	39,691	0.79	0.76	0.86	0.90
gpt-4.1-PC	27,175	0.74	0.76	0.79	0.83

Table 7.3: Model Performance

rephrasings were created, either manually or with the help of an LLM. The variations of a query must remain sufficiently close to the base query for the corresponding ground-truth run to remain valid and applicable. To quantify semantic distance, we encode each query using a Sentence Transformer model [Reimers and Gurevych, 2025], then compute the cosine similarity between each variation and its corresponding base query. Only those variations with similarity below 0.85 remain to ensure sufficient linguistic diversity (appendix E.1). This methodology is grounded in prior work on embedding-based semantic similarity [Elekes et al., 2017, Rekabsaz et al., 2017]. The final dataset consists of 135 queries (27 base + 108 filtered variations) with an average similarity score of 0.78.

To understand the reliability and efficiency of the system, we must analyze the entire trajectory taken to arrive at a result, including intermediate decisions, tool usage, and communication between agents. To ensure consistency and reproducibility, we apply a reset mechanism before each evaluation run based on a routine that resets the data lake to a predefined state before

Model	NEM ↑	NJS ↑	NLCS ↑	NED ↓	NUS ↑	NES ↑	TEM ↑	TJS ↑	TLCS ↑	TED ↓	TUS ↑	TES ↑
llama3.3:latest	0.02	0.19	0.14	0.86	0.30	0.31	0.02	0.13	0.11	0.90	0.31	0.32
finetunedllama:latest	0.04	0.21	0.19	0.81	0.34	0.36	0.08	0.16	0.15	0.85	0.36	0.36
qwen2.5:72b	0.00	0.34	0.22	0.78	0.44	0.45	0.07	0.21	0.18	0.83	0.47	0.47
finetuned-qwen:latest	0.00	0.37	0.25	0.75	0.50	0.52	0.09	0.29	0.24	0.77	0.53	0.53
command-a	**0.71**	**0.93**	**0.88**	**0.12**	**0.95**	**0.96**	**0.63**	**0.84**	**0.80**	**0.21**	**0.96**	**0.96**
gpt-4o-mini	0.38	0.76	0.67	0.33	0.83	0.83	0.41	0.63	0.59	0.41	0.84	0.84
gpt-4.1-mini	0.56	0.88	0.84	0.17	**0.95**	**0.96**	0.53	0.80	0.75	0.25	**0.96**	**0.96**
gemini-2.0-flash	0.53	0.73	0.69	0.31	0.75	0.75	0.48	0.63	0.60	0.40	0.75	0.75
o4-mini-high	0.23	0.84	0.66	0.34	0.92	0.93	0.19	0.72	0.54	0.46	0.95	0.95
gpt-4o	0.53	0.81	0.75	0.25	0.85	0.85	0.50	0.68	0.67	0.33	0.86	0.86
gpt-4.1	0.63	0.89	0.82	0.18	0.92	0.92	0.53	0.73	0.71	0.29	0.93	0.93
claude-3-7-sonnet	0.64	0.91	0.83	0.17	0.92	0.93	0.53	0.74	0.72	0.29	0.93	0.93
gemini-2.5-pro-preview	0.56	0.88	0.80	0.20	0.90	0.91	0.44	0.72	0.67	0.33	0.91	0.91
gpt-4.1-PC	0.47	0.81	0.71	0.29	0.84	0.85	0.37	0.61	0.58	0.42	0.85	0.85

Table 7.4: Node (**N**) & Tool (**T**) metrics across all models. Metrics marked with ↑ are better when higher; ↓ are better when lower.

queries are executed.

Token usage (**TOK**) denotes the average number of tokens used during a run. It reflects the verbosity and efficiency of the system and correlates with inference cost.

We use two metrics to assess whether a task was completed successfully.

ACC is a manually annotated accuracy score. Each run is labeled as either correct or incorrect, whereby the individual reasoning is also checked.

LACC serves the same purpose but is computed automatically by prompting a capable LLM (GPT-4o in our case) to judge the correctness of the answer only. While this approach enables large-scale evaluation, it may not match the precision of human annotations.

The **Pass@k** metrics assess whether the system produced the correct tool calls or code snippets and whether they succeeded within k attempts. We do not apply metrics for evaluating code generation (CodeBLEU/BERT [Ren et al., 2020, Feng et al., 2020]), because we prioritize tool invocation over code synthesis since generated code is short and tightly scoped to individual subtasks. Correctness is thus more effectively captured through the Pass@k and sequence metrics.

Node sequence evaluation focuses on the agent nodes visited during the reasoning process. We compare the predicted node sequence to a ground truth reference; both sequences are extracted from the messages of the run and ground truth example, where the order of messages determines the order of nodes in the sequence.

For **Tool sequence evaluation** we apply the same set of metrics and again compare to the ground truth sequence, which is extracted by collecting the name of each tool call that was made. The sequence for the run to be evaluated is constructed by iterating over the messages: if the message's `source_node` is `"tool_agent"`, the tool name is added; if it is `"code_agent"`, tool names from the example are added if they appear in the message content, otherwise a

placeholder is used. Sequences are evaluated with the following metrics:
Exact Match (EM): Binary match score (1 if identical, 0 otherwise).
Jaccard Similarity (JS): Ratio of the intersection to the union of predicted and reference nodes.
Longest Common Subsequence (LCS): Relative length of the longest common subsequence.
Edit Distance (ED): Normalized Levenshtein distance, giving 0 for identical sequences and 1 for completely different ones.
Unmatched Steps (US): Measures how many steps in the predicted sequence do not align with the reference via $1 - \frac{u}{L_{\max}}$, where u is the number of unmatched steps and $L_{\max}$ is the maximum sequence length set to 200 across all runs of varying lengths.
ES (Extra Steps): Measures how many extra steps e were predicted compared to the reference, via $1 - \frac{e}{L_{\max}}$.

All step-based metrics are normalized to 1, making them comparable across runs and tasks of varying lengths. For evaluation, we report average scores computed over all queries and runs. This enables robust comparisons across models and configurations. Together, these metrics provide a comprehensive view of system performance capturing not only whether the system gets the right answer, but also how it arrives at that answer. The selection of models is based on those capable of tool calling, a necessary feature for our system. Tables 7.3 and 7.4 summarize the evaluation results across all models. The first five entries correspond to open-source models including fine-tuned versions (see section 7.5.4). While LLaMA and Qwen are fully open-source, we treat Command-A as a mid-point: not fully open-source, but available for local inference unlike API-only models. The remaining models are closed-source and include offerings from OpenAI, Anthropic (, which requires a custom parsing mechanism), and Google DeepMind. Models are roughly ordered by parameter size. The final row shows results for our prompt compression approach using GPT-4.1 (see section 7.5.4).

Discussion of results

The results show that 70B open-source models, such as Llama 3.3 and Qwen 2.5, are approaching their limits for complex, multistep, and long-context tasks. While Qwen consistently outperforms Llama across most metrics, its accuracy remains insufficient. The marginal improvements from finetuning indicate that the current model scale and/or the available finetuning dataset are not enough to bridge the gap to production readiness. Qualitative inspection reveals frequent hallucinations and schema violations, particularly in scenarios involving long contexts, such as selecting objects from cache or synthesizing outputs from large JSON structures, even with larger models.

There is a notable performance jump between the 70B-class models and Cohere's Command-A model (111B), which ranks among the top three overall in terms of accuracy and achieves the best scores across both node and tool sequence metrics. Additionally, Command-A attains the highest pass@1 and pass@2 rates, indicating that its tool calls and generated code are the least likely to fail. These results underscore the benefits of scaling up model size and improving pretraining quality, particularly for structured, tool-driven multi-agent workflows.

Closed-source models consistently outperform the smaller open-source models across all evaluation metrics. The highest accuracy was achieved by `o4-mini` (0.90) with a `high` reasoning effort. However, this setting results in significantly higher token usage (50,466 tokens on average) and slower inference times, making it unsuitable for interactive applications where quick responses are expected. In contrast, models such as GPT-4.1 and Claude 3.7 Sonnet offer a strong balance between accuracy and efficiency. Notably, GPT-4.1-mini provides a cost-effective option, maintaining high performance while reducing resource consumption compared to its larger variants. A general trend is observed where models with higher accuracy also consume more tokens and vice versa. This is likely due to their ability to complete more steps, generate more detailed reasoning, and avoid premature failures.

Despite reducing the average token count, applying prompt compression led to a significant drop in accuracy and other metrics. This is primarily because many critical prompt elements (e.g., function signatures, object cache, user queries, API responses) cannot be compressed without introducing errors.

A positive finding is that LLM-based accuracy (LACC) closely matches human-annotated accuracy, suggesting that LLMs can reliably self-assess the correctness of their outputs in this setting.

In our setting, finetuning open-source models led to only marginal improvements. This could be due to factors such as limited dataset size, model capacity, or the specific finetuning strategy used. Given both annotation effort and compute cost, the current cost-benefit trade-off limits the practical value of finetuning. Again, model scale and pretraining quality remain the most important factors for accuracy and reliability.

Overall, accuracy remains the decisive metric for judging model readiness in our multi-agent setting. As shown in Table 7.3, open-source models achieve limited accuracy (0.12–0.33), even when fine-tuned, confirming that they are not yet competitive for complex, multistep workflows. In contrast, proprietary models consistently surpass 0.70, with GPT-4.1, Claude 3.7 Sonnet, `o4-mini-high`, and Cohere's Command-A all achieving accuracies of 0.87 or higher. Among them, Command-A, a relatively small model (111B) matches GPT-4.1 at 0.87 and is only slightly outperformed by Claude 3.7 Sonnet (0.89) and o4-mini-high (0.90). This demonstrates that top proprietary models are currently necessary for reliable execution in our setting, while open-source alternatives remain behind. Notably, the models with the highest accuracy also achieve the best `pass@1` and `pass@2` scores, meaning they converge faster and with fewer failed attempts.

Beyond accuracy, the node and tool sequence metrics reveal important qualitative differences between models. Command-A not only achieves a high accuracy but also shows good alignment with ground-truth sequences (NEM 0.71, TJS 0.84), indicating that its reasoning process closely mirrors the intended workflow. This reflects Cohere's training choices, which emphasized extensive instruction-tuning and tool use with the explicit goal of reproducible, API-like outputs for enterprise use cases rather than open-ended free-form text generation.

By contrast, open-source models often deviate substantially from the correct node and tool order, even when they occasionally produce the right final answer. This suggests brittle

reasoning and low robustness to linguistic variation. GPT-4.1 and Claude 3.7 balance high accuracy with strong sequence metrics, showing that they can both reach the correct result and follow a plausible reasoning trajectory. Interestingly, token-heavy models like o4-mini-high achieve strong final answers but weaker sequence alignment, implying that sheer verbosity can compensate for reasoning mistakes. It often deviates from the expected path, with extra steps, repeated tool calls, or slightly different orderings. Instead of following a direct route, the model works through several alternatives until it arrives at the correct result. These results underscore the importance of evaluating not just correctness but also reasoning consistency, efficiency, and robustness, which are critical for multi-agent systems where interpretability and reproducibility matter. Overall, the quantitative evaluation revealed recurring failure modes, which in parts are further investigated in the following sections:

- **Retrieval errors.** Incorrect or incomplete functions were sometimes retrieved.
- **Cache mismatches.** In longer workflows, references to cached objects could desynchronize, causing agents to operate on outdated or missing instances.
- **Verbose reasoning.** Some models (e.g., `o4-mini-high`) reached correct answers through excessively long trajectories with redundant tool calls or detours, reducing efficiency and interpretability.

7.6.2 Qualitative Evaluation.

We qualitatively assessed the system across a range of real-world and challenging scenarios highlighting both successful outcomes and common failure modes. The three real-world datasets are based on Industry 4.0 tasks, which we also used for evaluating AutoML approaches [Hoseini et al., 2024c]. We tested for successful pipeline generation (understand the task, upload a dataset, understand the data, design AutoML experiment and generate a notebook with corresponding code, forward the user to corresponding page) and do not compare ML-related performance metrics:

1. **Object Detection:** The dataset in [Hoseini et al., 2022] comprises 3D height profile images of coated surfaces with bounding box annotations for defect detection. We tested the query: *"Ingest the dataset object_detection.zip. Create a machine learning notebook with object detection for it."* Successful execution required dataset creation with correct parameters, ingestion, experiment setup, and generating an AutoML notebook for object detection. GPT-4.1 failed at several steps, including unnecessary dataset searches, incorrect parameters, and wrong component calls. Gemini 2.5 Pro also introduced errors and skipped experiment creation, however, when the prompt was clarified, it completed all key steps correctly.
2. **Image Segmentation:** The dataset [Zhang et al., 2022] contains crosscut test images with pixel-wise segmentation masks for delamination detection. The tested query was: *"Ingest segmentation.zip to create a dataset. Create a machine learning notebook with image segmentation for it."* Successful execution required to define the dataset with appropriate ingestion parameters (such as `read_type = IMAGE`), ingest the data, create a ML experiment, and generate an AutoML notebook for image segmentation. Gemini 2.5 Pro completed all steps

correctly, including generating a valid dataset definition, specifying image and mask fields, and providing a working notebook link, while also informing the user about potential ingestion delays. GPT-4.1 omitted experiment creation and used incorrect parameters, but succeeded when given a more explicit prompt specifying the dataset structure and task.

3. **Tabular Regression:** This dataset [Hoseini et al., 2024e] contains structured sensor readings from automated scratch resistance tests, used to predict the force for breaking through coatings with a needle. The tested query was: *"Use pda_2023-04-18_10-13-22.csv to create a dataset about surface coatings data. Ingest and publish it. Perform tabular regression to predict the* $x_position$ *where the needle breaks through the coating."* Successful execution required defining the dataset with correct parameters, ingesting and publishing it, creating an experiment, and generating a regression notebook with the right target. Gemini 2.5 Pro performed all steps correctly; although it initially used the wrong function for notebook creation, it recovered by switching to the appropriate API and ultimately provided a working notebook link. In contrast, GPT-4.1 failed to complete the workflow, adding unnecessary steps and not generating the notebook.
4. **Robustness to unusual queries:** When asked to provide an unavailable file the system attempted to process a missing dataset and ultimately returned a helpful error message, though it could not validate file existence in advance.

   When asked to perform reinforcement learning, which is not supported, the system correctly executed ingestion and experiment setup but failed at notebook creation, indicating a lack of capability awareness.

   A request to compare coating data with NASA's lunar soil database was decomposed into subtasks but failed gracefully with an explanation of missing data sources.

   For a nonsensical request about cars in a garage, the system attempted a literal interpretation before issuing a fallback, highlighting robustness but limited commonsense reasoning.

   When asked about the weather on Mars, the system attempted unrelated steps before acknowledging the impossibility, showing polite but delayed rejection.

Overall, the qualitative evaluation highlights both strengths, i.e., graceful failure handling, low hallucination rates, and current limitations, i.e., the lack of intent clarification and proactive user guidance. Across all cases, the system reliably decomposed user queries into subtasks and generally avoided hallucinations, returning clear fallback responses that explained why a request could not be fulfilled. At the same time, infeasible requests were usually only rejected after partial execution rather than being filtered upfront. As in the quantitative setting, prompt specificity strongly influenced outcomes: explicit instructions often enabled correct multistep execution, while vague prompts led to partial failures or omissions. Together, the quantitative and qualitative results demonstrate that evaluating NLIs requires going beyond correctness, towards robustness, efficiency, and reasoning transparency.

Decomposed Subtasks	Top Retrieved API Function
Search for the dataset Sensor_Data	datasets_search()
Get all attributes and identify failure-related attributes	get_all_attributes()
Create an ML experiment for predicting failures	create_experiment()
Create a semantic model targeting failure prediction	create_modeling()
Create a machine learning notebook	create_ml_notebook()

Table 7.5: Task decomposition (shortened) for the query *"Predict failures in the Sensor_Data*" and retrieved functions

7.6.3 Impact of Retrieval Quality.

The effectiveness of the agent framework depends heavily on the quality of retrieval results. In practice, we observed reliable retrieval performance when user queries are successfully decomposed into atomic subtasks by the query decomposer, as the clarity of a subtask directly determines the relevance of the retrieved functions. Hence, the query decomposer is of critical importance and particular attention was devoted to its design to ensure the agent is explicitly aware of available API functionality for arbitrary user queries. For example, consider the following trace, in which the query decomposer generates the subtasks shown in table 7.5 for the user query: *"Predict failures in the Sensor_Data"*. Even though the user query is fairly vague, the agent identifies most of the required steps correctly but also introduces a superfluous action (*"Create a semantic model ..."*). The subtasks in table 7.5 are then expressed clearly enough to ensure that retrieval for each of them consistently returns relevant and accurate results. Nevertheless, the successful execution of more complex workflows remains highly dependent on the specificity and clarity of the initial user prompt.

7.7 Conclusion

This chapter examined how LLM-based, multi-agent systems can serve as an intelligent interaction layer for semantic data lakes, with a particular focus on empowering non-technical users to interact with complex data management and analytics infrastructures.

With respect to RQ4, the proposed natural language interface demonstrates that complex data-management tasks can be successfully executed through goal-oriented natural language interaction, without requiring users to engage with code, query languages, or system-specific APIs. The results show that semantic context, retrieved API knowledge, and structured agent coordination can effectively translate user intent into executable operations, thereby transforming semantic data lakes into genuinely self-service systems for domain experts.

Regarding RQ2, the NLI illustrates how semantic data lake operations can be automated and exposed through intelligent interfaces that abstract over heterogeneous storage systems, execution engines, and data modalities. Concerning RQ3, the multi-agent architecture provides evidence that a hybrid of retrieval-augmented reasoning, constrained tool invocation, and

iterative validation can reliably orchestrate multistep workflows across SEDAR's components. The evaluation further reveals that capable LLMs can handle such tasks with high reliability, whereas current open-source models remain limited, thereby quantifying both the strengths and boundaries of LLM-based automation in semantic data lakes.

Despite these promising results, several limitations remain. Tool invocation remains highly dependent on retrieval quality; errors in retrieving method descriptions or object references can propagate through entire workflows. Long agent chains introduce fragility in the object cache, and smaller models struggle with long-context reasoning and structured outputs. Computational overheads arise from re-ranking stages and large reasoning models, especially under constrained hardware. Prompt compression reduces token usage but currently harms accuracy, and fine-tuning only partially mitigates these challenges. Moreover, the system decomposes user queries only once, without adapting to intermediate results or user feedback. Introducing adaptive planning and intent clarification mechanisms would improve robustness and user experience.

Positioned within the broader evolution of SEDAR, from a classical data lake to a semantic data lake and ultimately toward an AI-assisted data fabric, this chapter shows how natural language interfaces can remove the final accessibility barrier between complex data infrastructures and domain experts. As LLMs, retrieval pipelines, and reasoning-oriented architectures continue to mature, such interfaces are likely to become standard entry points for large-scale, heterogeneous data ecosystems.

Ultimately, the contribution of this chapter lies not in the specific implementation of SEDAR's NLI, but in the methodology it demonstrates: a reusable architectural and interaction pattern for translating natural language intent into structured, tool-grounded actions over heterogeneous data-management and analytics systems.

8

Applications of SEDAR in Data-Driven Chemistry

„Die Eule der Minerva beginnt erst mit der einbrechenden Dämmerung ihren Flug.“

“The owl of Minerva begins its flight only with the onset of dusk.”

— G. W. F. Hegel

This final chapter illustrates how the concepts and systems developed throughout the dissertation operate in practice. The datasets presented here stem from a series of applied ML studies at the HIT Institute and provide a common basis for evaluating SEDAR across its core components.

We proceed in three steps. First, we introduce the chemistry and coatings datasets and show how they are used to assess the AutoMLWrapper and the agent-based workflows. Second, we demonstrate an end-to-end ML process in which semantic modeling and OBDA support automated analysis in a new chemistry use case. Finally, we revisit SEDAR through the lens of the DATER framework to position the system within the broader landscape of data architectures.

This chapter is based on the publications [Hoseini et al., 2022, 2024e, 2025a, 2024c, 2026b, Zhang et al., 2022].

Chapter Outline

This chapter is structured as follows:

- Section 8.1 — Introduction of the research environment situated within coating and paints chemistry of series of applied ML studies at HIT, which form the basis for evaluating automated ML and agent-based workflows. Each applied ML study resulted in a dataset:
  - Section 8.2 — Object detection dataset on 3D height maps for identifying coating defects, supported by automated imaging and data handling.
  - Section 8.3 — Image segmentation dataset for automating cross-cut characterization in coatings, including domain setup and preprocessing.
  - Section 8.4 — Regression dataset of scratch resistance from manufacturing sensor data, illustrating predictive modeling in a smart Industry 4.0 pipeline.
- Section 8.5 — Systematic evaluation of the AutoMLWrapper and the KG-enhanced Data Science Agent using these datasets.
- Section 8.6 — A semantic end-to-end ML workflow integrating ontologies, OBDA, and LLM-based code generation to solve a real chemistry use case.
- Section 8.7 — Evaluation of SEDAR using the DATER framework, positioning the system within the broader architectural landscape.
- Section 8.8 – Concludes the chapter by assessing the contribution to with regards to the research questions (section 1.2).

8.1 Introduction

Chemical engineering in materials science involves tasks such as formulation and transformation of raw ingredients into intermediate products, synthesis of materials, and the final application of these materials. Commonly formulated products include polymeric materials like plastics, composites, coatings, or adhesives. To advance in this domain, an understanding of the relationship between chemical compound properties and structure of all ingredients (resins, polymers, solvents, additives, and inorganic and organic particles) is required, as well as knowledge of the physical parameters governing material properties and processes.

Automated measurement of surface integrity is an important theme that has frequently reoccurred in cooperations with regional partners at the HIT. Organic coatings are commonly applied to protect working surfaces from corrosion, abrasion, and erosion [Streitberger and Goldschmidt, 2018] or impart an aesthetic appearance [McKnight and Martin, 1998]. These coatings act as barriers covering the underlying substrate from the environmental impact and promise long-life durability. The most critical failures are caused by delamination or other surface defects and result in malfunction of the workpiece [Pappas, 1989, Fotovvati et al., 2019].

Transitioning from traditional qualitative evaluations, there's a growing emphasis on deriving precise, quantifiable data. Automation is the key catalyst for this shift. Rather than relying on generalized assessments, state-of-the-art scratch technologies [Steinmann et al., 1987, Ran-

dall, 2019, Lu et al., 2011, Bull and Berasetegui, 2006], coupled with ML algorithms [Lim et al., 2022], have ushered in a paradigm where adhesion failures are delineated with exact measures. However, automation frequently encounters obstacles due to a fragile optical evaluation that proves overly sensitive to changes in the experimental setup [Zhang et al., 2022, Polke et al., 2023].

The goal of the research at the HIT and in particular the i²DACH-project (section 1.1) is the use of ML to accelerate and improve chemical research, development, and manufacturing by leveraging data to make predictions, discover patterns, and optimize processes. For this, on the one hand, SEDAR manages the heterogeneous process data; on the other hand, it not only manages the ML methods used to analyze this data, but also orchestrates the automation of the HTF system across multiple process levels and integrates ML directly into the plant's control layer (see figure 1.1a). Throughout the course of this PhD project, numerous incremental steps have been taken toward this vision. In the chapter, we present selected applied research efforts.

Datasets A major challenge in data-driven chemistry is not just the modeling itself, but also the acquisition of high-quality data under controlled, reproducible conditions with minimal variance. The first two sections in this chapter (sections 8.2 and 8.3) present two such approaches in which newly acquired measurement equipment is utilized to yield high-quality datasets for the analysis of the surface integrity of coatings.

Section 8.4 presents a first holistic approach in which a regression dataset emerged. It not only covers the automated preparation, characterization, and data acquisition for coating samples, but also integrates a deep learning model for evaluating scratch resistance directly from sensor data. Crucially, this approach implements a feedback mechanism in which the analysis results are fed back into the system enabling adaptation of formulations and experimental parameters in future runs. This integration of data acquisition, ML-based evaluation, and automated execution represents a concrete step toward closing the loop between experimentation and process optimization as originally envisioned in the project proposals (see figure 1.1a).

Sections 8.5 and 8.6 then describe based on the previous three datasets, concrete approaches on how to automate the process of ML modeling, but also data preparation and integration with LLMs.

8.2 Object Detection on 3D Height Maps to Identify Coating Defects

This section is based on [Hoseini et al., 2022].

Protection of the substrate is the most important task for paints and coatings in many applications. If defects occur [Kornum and Nielsen, 1980, Dietrich, 2021], caused by the application process or inappropriate lacquer formulation, the barrier effect fails and coatings lose their decorative appearance. The knowledge about the specific causes (like pigment aggregates,

foreign particles, outgassing of air or solvent resulting in contaminations and pinholes [Kornum and Nielsen, 1980, Dietrich, 2021]) is significant for adjusting the formulation and the process parameters. Identifying and classifying defects in an evaluation process of coatings material is usually done manually by the human observer. Analytical methods may help to disclose the reason for the occurrence of defects on coatings. Techniques like cross-section polish [Schoff and Associates, 2006], microscopy [Solomon and Solomon, 2018, Spring et al., 2008] are useful to narrow down the origin of the failure if the optical inspection or a 2D microscopy image is inadequate. 3D imaging of the profile of the defect often reduces time and effort deriving the type of the cause by its shape [Schneider et al., 2001, Zhou et al., 2019, Ma and Lee, 2022]. All these methods support the categorization and make automatic identification accessible, e.g., for process control in coating applications [Li et al., 2021, Kieselbach et al., 2019].

8.2.1 Sample Preparation

For the preparation of samples several coatings and substrates have been combined creating a data set covering four different defects, where the coating was applied to the substrates via draw-down or via spraying. In order for the model to better generalize to unseen data, it is advantageous to create a diverse dataset that contains a high degree of variance in the patterns it aims to learn. Therefore, we used four types of coatings with different colors on steel substrates.

Several classes of defects were generated. Only defects with a similar circular shape were considered for this study to demonstrate the advantages of training on height information obtained from the laser scanning microscopy when compared to a usual (optical) microscopy image. The defects were caused by improper handling and recipe adjustments to simulate a real case:

- **craterings**: silicone impurities on substrate, overdosing flow additive
- particle **contamination** (also as known speks): insufficient milling of the pigmented lacquer
- **pinholes**: insufficient addition of benzoin of the powder coating
- **blisterings**: storage of the hardened blue top coat in water

Depending on the substrate, coating material and film thickness the appearance and defect density differed between the test panels resulting in a variety of features, which the algorithm can be trained on. Because each image from the microscope contained several defects, a total of 8 substrates was sufficient. All coated substrates were allowed to dry at room temperature with a relative humidity of (50 ± 5) for two weeks. For all samples the achieved dry file thickness lies between 40 and 60 μm based on different types of coatings.

8.2.2 Automated imaging

Due to the limitation of human eyes, humans need to observe the surface on multiple angles to gather 3D information of the surface in order to identify the type of the surface defects. As described earlier, this process is rather complicated, strongly relies on experience and is still error-prone. A traditional microscope usually only captures the 2D information at one angle,

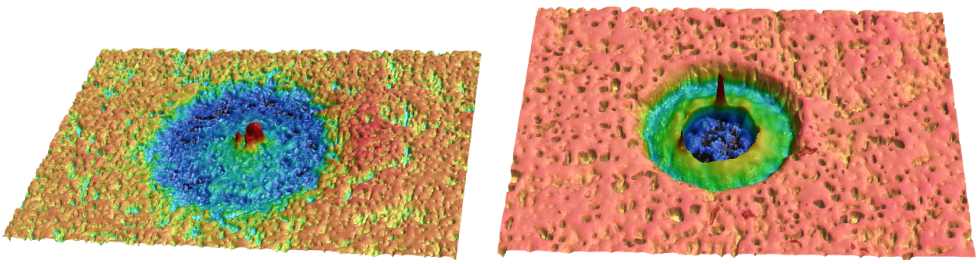


Figure 8.1: Comparison of the height profile of craterings and pinholes.

where the image quality can also be greatly influenced by external conditions, such as color or lightning conditions. Knowledge about the 3D structure of a defect would enable much better discriminative power. A pinhole for example, is defined over deep holes with steep walls, which is straightforward to recognize when there is a height profile at hand, but difficult to derive from an optical image only. Figure 8.1 shows how craterings and pinholes look similar in 2D, but very different in 3D. The left defect is found to be a cratering through the detection of low surface residuals in the center. The right defect is found to be a pinhole, which are caused by the ebullition of the solvents. The defects have similar structures and are difficult to be distinguished by 2D microscopes.

A considerable amount of literature has been published on 3D-object reconstruction of the surface from multiple images at different angles, but the achieved accuracy is still limited in regards to the macro-object and there is a significant computational cost involved ([Tomasi and Kanade, 1991] [Xiang and Sheng-yong, 2011] [Choy et al., 2016]). With the development of modern laser technology, a laser-scanning microscope provided by *Keyence* can capture the 3D information of the surface with high precision and robustness [Colaço et al., 2012] independent of the lighting condition and in comparison to the 3D reconstruction approach, it is much faster and easier to operate. In this work, for each sample, both the RGB images and the 3D depth image are captured with an optical/laser hybrid microscope, but only the 3D depth images are used for the detection of defects. The optical images are reserved for later presentation to the user.

An *automation script* is capable to navigate and scan a sample starting from a center point and capture a user-defined number of data points on a grid. In order to cover as much as possible from the sample, the lowest available magnification factor is selected, such that each data point contains an area of approx. $2076\mu m$ by $2768\mu m$ represented by three images of 1024×768 pixels: optical image, depth visualization and raw depth data (see Figure 8.2). On the left you see the optical snapshot, the height image (center) is a visualization generated by proprietary software provided from the manufacturer *Keyence* and the right image was generated by stacking the raw height data to artificially construct an RGB image, see section 8.2.3 for details.

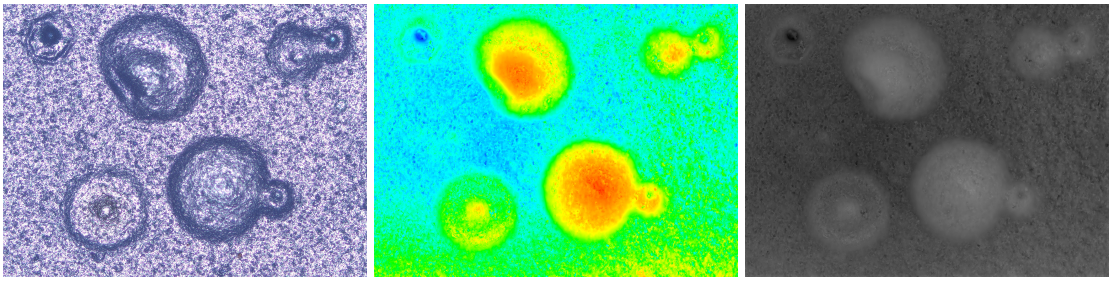


Figure 8.2: Output images of the automation script.

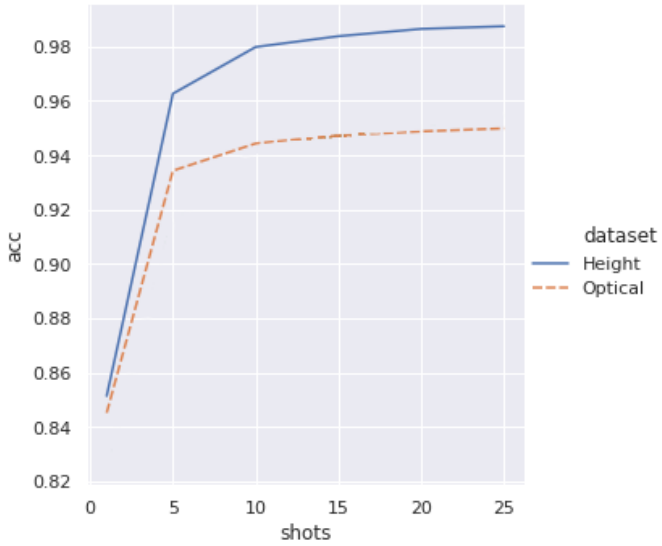


Figure 8.3: Accuracy of a classification model [Doersch et al., 2020] plotted against the number of available samples (shots) when evaluated by a classification model.

8.2.3 Data Preprocessing

While the RGB images are outputted by the library in JPEG- and BMP-formats, the raw height data is stored in a proprietary format (.vk6) and has to be converted first using an open-source software called *Gwyddion*[1]. Because the raw data is composed of a single channel, basically a matrix storing the height measurements in μm in each of the 1024x768 cells, it was necessary to artificially construct an RGB image, since most algorithms for vision are designed for optical data. For this reason, we repeat the raw data three times and convert it artificially into RGB, the right image in Figure 8.2 illustrates an example for such an image.

In early initial experiments aiming at the classification of different defects using a different model [Doersch et al., 2020], we found that the height profiles achieve much better performance, as can be seen in Figure 8.3: The blue line represents the artificial RGB images, generated from the microscope's raw data. The red line represent optical images. This early study showed that the height profile increases the discriminative capabilities of the model compared to optical images.

[1] http://gwyddion.net/, last access: 06.04.2022

Finally, we store the data in a binary representation. Each image is accompanied by its label, containing ground truth information about the class and the location within the image. The goal for the model is to reproduce these on a novel previously unseen image given its corresponding height profile and additionally output its confidence about the detection. The dataset is divided into a training, and testing set, where each data point may contain one or more objects; for a detailed division:

- 315 training points, where each data point consists of a stacked height profile with a resolution of 1024 by 768 (by 3) and the corresponding labels. Out of this training set we randomly selected 70 data points to train the ML model.
- 75 data points were kept to form a test set. The set is composed of 15 data points per class plus 15 data points that contain multiple distinct classes in the same image.

8.3 Image Segmentation for Automating Characterization of Cross-cut Tests for Coatings

This section is based on [Zhang et al., 2022], where G. Zhang had the primary role in the experimental research and S. Hoseini wrote the paper.

One important procedure for rating the destruction of coatings is the so-called Cross-cut test [ISO2409:2020, 2020] evaluating the delamination of coatings by comparing the pattern of the damage being closely related to the delaminated area. Previous work has targeted a quantitative evaluation of the test [Zhao et al., 2011], however automating the suggested method is not feasible and computational approaches for image segmentation are potentially more efficient. Online imaging is a fast quality control, that is already used in post-processing of industrial manufacturing [Liao and Lee, 2016, Bongardt et al., 1999]. An automated image recognition valuing the defect of the Cross-cutting would be desirable for a faster throughput with no interference of the human operator. The differences between intact surface and the damage need to be recognized even in case of small optical deviations between some combinations of substrates and coatings (e.g., clear coating on glass substrate).

We focused on the automated evaluation of delamination by cross cutting. The cutting itself was performed by an automated system handling incoming test panels with several kinds of soft and hard material as the substrate. The pattern of damage was imaged and evaluated by an ML algorithm segmenting delaminated and intact area.

8.3.1 Sample Preparation

For the preparation of samples several coatings and substrates have been combined creating a data set covering all possible categories of the cross-cut test, where the coating was applied to the substrates via scrambling. In order for the model to better generalize to unseen data, it is advantageous to create a diverse dataset that contains a high degree of variance in the patterns it aims to learn. We used four types of coatings with diverse colors and degrees of gloss as well as four types of substrates (steel, electrogalvanized, phosphated, and glass) to

create samples with differing appearances.

Automated cross-cutting The cross-cut and imaging have been carried out with an automated high-throughput equipment, that handles coated test panels and cuts, scratches or brushes the surface, controlled by an embedded computer (see figure A.2). The device was set up comparable to other automated platforms processing such destructive mechanical testing being available off-the-shelf [Erichsen GmbH & Co. KG, 2021]. A rail-system moves the panels from the storage location to the testing and imaging station. Two z_1, z_2-linear servo drives with clamping systems can be mounted with tools like a wolfram needle for cutting, and a brush to remove delaminated material. The pattern of the cut and reciprocal strokes with the brush removing delaminated coat were accessible due to the maneuverability of the automation. Furthermore, the force resulting in a pressure of the two tools on the surface was controlled during the movement and achieved comparable destruction regardless of the hardness of the coated surface. The controlled force of the tools also allows mechanical tests like the scratch resistance [BSI Standards, 2011] that require different loads identifying the point of material failure. All samples for this study were first cut with a force of five Newton and then brushed with a force of eight Newton. Three cross-cuts were carried out on each panel for the evaluation of the delamination. The imaging was performed with an optical setup for a balanced illumination of the sample avoiding artifacts from reflections or other side effects.

8.3.2 Data Preprocessing

The dataset contains 217 images, which have been taken with a camera under constant conditions and with no further changes to the camera and cross-cut test parameters, to ensure objectivity of the process. The images have then been cropped to contain only the 'cutted-on' area, where the grid has been applied to the substrate. Subsequently, a graphics editor was used to define the ground truth as a binary mask, i.e., manually labeling each pixel location with a value of 255 for areas in which there is no coating left and 0 for pixel locations that still show a coating on the surface. In addition, the image is transformed from RGB color space into grey scale. The manual segmentations have been used as the ground truth in both segmentation model training and final segmentation performance evaluation. This process is illustrated in figure 8.4, where the center and right picture form one of the input data points for the training of the algorithm.

The goal of the model is to reproduce the existing and non-existing coating locations for each pixel when given a new, previously unseen sample image. To relate to DIN EN ISO 2409 [9], the percentage of the remaining coating was calculated to characterize the adhesion strength of a certain sample in an automated manner. There is large consent that successful training of deep networks requires many thousand annotated training samples [Goodfellow et al., 2016]. While this may be true for the traditional classification tasks, where you have one target for an entire image, it is different for segmentation tasks. Here, each pixel location of the image serves as a target (either 0 or 255), which is why the network can adapt much easier as

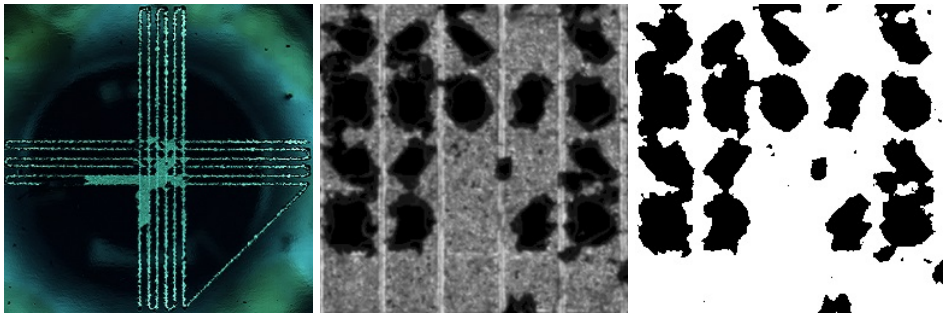


Figure 8.4: Segmentation images and mask: (a) original image (b) cropped image (c) manually labelled image

there is more ground truth available to learn from [Ronneberger et al., 2015]. The dataset is divided into a training, validation, and testing set:

- 174 training points, where each data point consists of a grey scale image with a resolution of 256 by 256 pixels and a binary mask of the same resolution,
- 35 data points for validation during training,
- the remaining 8 data points were kept to form a test set combined with 23 selected points from the validation set to contain 31 points for evaluation.

8.4 Regression of Scratch Resistance from Sensor Data in a Smart Manufacturing Pipeline for Coatings

This section is based on [Hoseini et al., 2024e], where we present an integrated system composed of the automation platform at the HIT (section 1.1), a characterization module to evaluate the material, and a deep learning model to accelerate the development of new coatings by evaluating their scratch resistance using sensor data independent of images. This integration showcases the interplay between chemistry, automation, and data science as a basis for smart manufacturing.

An essential step towards fully automated chemical processes through data-driven models involves automating sample characterization as part of a comprehensive ML process. A modular automation framework for data acquisition, data processing, chemical process modeling, and its optimization utilizing ML is demonstrated by Polke et al. [2022, 2023]. Major challenges in terms of hardware interfaces, interoperability, security, data management and adaptation of the process control system (PCS) are to be addressed. With the increasing interconnection of machines, not only the volume but also the variety of data is being expanded. The data life cycles of collection, processing, combining, analyzing, and feeding the results back to the machines are becoming increasingly challenging for smart manufacturing processes to complete, because data management is not continuous.

Related Work In [Polke et al., 2023], a modular automation framework for optimizing coating formulations through ML is presented. There, the evaluation of the coating's scratch resistance

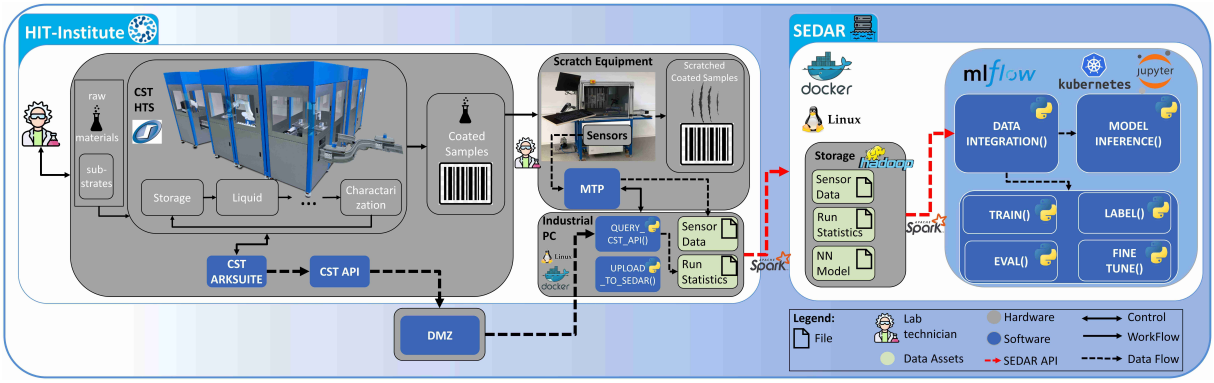


Figure 8.5: Schematic diagram of the workflow from left to right

was based on optical images. This turned out to be instable when confronted with varying coating colors, light conditions and substrate types. In contrast, this work is independent of images and instead relies on real-time sensor data to improve robustness against environmental changes. Further, our aim is to present a coherent automation workflow.

The fusion of materials science and modern digital technologies, including robotics, cloud computing, and artificial intelligence is an ongoing topic of research. There exists many proposals to incorporate artificial intelligence for replacing traditional manual, analog and human-intensive work in chemistry [Pyzer-Knapp et al., 2022, Zhang et al., 2023a, Butakova et al., 2021, Burger et al., 2020]. Various proposals demonstrate automation pipelines on how to incorporate modern cloud architectures [Li et al., 2020, Zhu et al., 2022] that can propose scientific hypothesis and generate experiment plans, execute the complete procedure (synthesis, characterization and performance testing) and build predictive models. The widespread adoption of robots for smart manufacturing in material science is challenging because of the diversity of sample types, operations, instruments and measurements required [Zeid et al., 2019]; thus handling the diversity and heterogeneity of data is crucial.

8.4.1 Pipeline Automation

The preparation of the coating samples is conducted via the HTF system[2] (see figure A.1) acquired by *Chemspeed Technologies AG*[3] (CST). This advanced robotic system offers significant advantages over manual processes. The use of automation ensures controlled and reproducible conditions, thus minimizing variance between samples and yielding high-quality data.

The process is illustrated in figure 8.5 progressing from left to right. The initial step, situated on the far left, commences with manual tasks carried out by lab technicians:

1. Substrate preparation
2. Module preparation (involving raw material handling, registering in *ArkSuite*[4], filling the

[2] https://youtu.be/aV1b44N69G0?si=bjGhIRK-xDAtlKwV&t=77, accessed on 02.05.2026

[3] https://www.chemspeed.com/, accessed on 02.05.2026

[4] https://www.chemspeed.com/software-and-digitalization/arksuite/, accessed on 02.05.2026. Workflow management software for orchestrating the automation platform.

liquid cartridges with raw material, standard checks for each module, etc.)
3. workflow definition for the HTF equipment on the CST ArkSuite software
4. determination of dosing parameters, typically involving preliminary experiments for calibration.

As discussed in section 8.4.2), a workflow comprising 12 samples is carried out. The HTF assigns identifiers as barcodes to each sample for future traceability.

Subsequently, the coated samples are manually retrieved from the storage module by lab technicians and transferred to the scratch equipment. The PCS, installed on an industrial PC, orchestrates the module and captures signal data. For the efficient integration of modules into a PCS, a semantic description of the modules and their services is needed. These requirements are defined in the standardization of the Module Type Package [VDI/VDE/NAMUR 2658 Part 1:2019-10, 2022].

Simultaneously, the PCS serves as the central conduit for outbound data transmission. To secure the highly sensitive HTF equipment, it is necessary to establish an isolated network to facilitate all communication for the several CST components. A Demilitarized Zone (DMZ) is configured to shield the CST API from incoming requests. This arrangement is essential due to the HIT functioning as an open innovation hub, with various vendors, including major firms, entrusting critical protected data to be stored in the CST database. The DMZ only permits authenticated users to query the CST API for data explicitly approved for public use. In our scenario, the PCS queries the CST API via the DMZ to retrieve the *run statistics*, a CST data object containing comprehensive information about the workflow, including machine parameters and characterization results. This information is vital for correlating the neural net's output to specific samples.

The PCS is most suitable to control machines and capture their sensor data. To manage numerous data assets rigorously over time we utilize SEDAR (chapter 4). The *SEDAR-API* (section 4.3.4), installed on the industrial PC, uploads the data assets, i.e.,*run statistics* as well as sensor data from the scratch module, to SEDAR. At this juncture, a data scientist, logged into SEDAR, accesses the data also via the *SEDAR-API* and integrates the run statistics with the sensor data. The trained neural network is loaded, and model inference is executed for each scratch individually. The run statistics allow to trace individual predictions back to the specific coated sample. We compute the average of ten predictions for a sample, extract the nominal force at the penetration point, and interpret this as a measure of scratch resistance.

The advantage of employing SEDAR lies in the centralized management of (meta-) data and integrated ML pipelines. Thus, all training, labeling, and evaluation scripts, model artifacts (versions, weights, architectures, hyperparameters, ...) as well as datasets are accessible, registered, and meticulously documented and available to anyone with corresponding access.

8.4.2 Experimental Setup

In the following the experimental setup as required for the executing the pipeline in section 8.4.1 is described.

Materials & Sample preparation The training samples are coated manually with a two-component urethane-based **top coating**. This coating was selected for its mechanical properties that can be adjusted through the ratios of its crosslinker.

For the test set, a two-component, high-solid, urethane-based **clear coating** is employed, where the coating's formulation is optimized to include only one crosslinker. The proportion of this crosslinker is varied, maintaining it between 34 and 66 wt% relative to the binder, to achieve increasing degrees of scratch resistance. These samples are prepared using the high-throughput automation pipeline (see figure A.1) marking a shift from manual to automated processing.

Automated Sample Preparation The sample preparation for the test set involved six modules:

1. Storage Module: An autonomous shuttle begins by fetching an empty beaker and a steel substrate from storage.
2. Liquid Module: The shuttle moves to the liquid module where an analytical balance under the shuttle monitors and controls the weight of the dispensed liquid to achieve target weight for precise formulation.
3. Mixing Module: The dispensed mixture is transferred to the mixing module where it is stirred at a speed of 1500 rpm for 3 minutes using a *Speedmixer DAC400*[5].
4. Application Module: The mixed coating is then moved to the application module. This module houses a spray cabinet with a vacuum chuck that moves in *y*-Z coordinates, a stationary spray pistol, and a multifunctional robotic arm. The substrate is first placed in the spray cabinet by the robotic arm and secured using the vacuum chuck from behind. The robotic arm then fits a syringe, extracting 8 mL of the coating using an embedded microdispenser. Then, a tapered dispense tip is attached to the syringe, which is then aligned with the spray pistol from the right, positioning the dispense tip directly in front. The coating is dispensed through the syringe and *in-situ* dispersed using compressed air from the spray pistol, while the vacuum chuck moves in a preprogrammed rectangular-shaped pattern to ensure a uniform application on the substrate. The benefit of this approach is that a contamination-free application can be ensured.
5. Heat Curing Module: The coated substrate is subsequently processed in the heat curing module. Initially, the robotic arm positioned the substrate in the flash-off location, where it underwent a flash-off period of 10 minutes. This duration, as well as the subsequent curing time, is automatically tracked and logged by the server, ensuring precise adherence to the protocol. Following the flash-off phase, the substrate is transported to an oven designed with a drawer-like structure, each drawer accommodating two substrate holders. The substrate then undergoes a curing process lasting 45 minutes at 80 °C.
6. Characterization Module: Subsequently, the coating's thickness is measured in the charac-

[5] https://hauschild-speedmixer.com/de/products/dac-overview/model-series-400-1100/, accessed on 02.05.2026

terization module, using inductive coating thickness[6] measurement equipment.

Lastly, both the coated substrate and the beaker are stored in the storage module.

Automated Scratch Testing At the HIT there exists an experimental setup for automated scratch tests shown in figure A.2. As part of the preparatory phase, a variety of coatings are applied to metal sheets within a chemical laboratory setting equipped with MTP. Coated samples are inserted into the testing equipment and fixed by vacuum. Following this, the metal sheet is moved below a needle which scratches lines in a defined way onto the applied coating. To evaluate the coatings scratch resistance, the forces and positions in x-, y-, and z- direction are transferred to an OPC UA Server. Thus, process control systems are able to acquire the process data during the scratch tests. The scratch module uses three synchronous motors to move in three directions of a Cartesian coordinates system. The metal sheet is located on a carriage which only moves in x-direction. Two force sensors are installed in this carriage. The movement in y-direction is carried out by the needle. The needle moves vertically in z-direction and applies the force on the coating, which is measured by the force sensors. A spring is used to control the force by adjusting the z-position of the needle.

Data Collection Using the previously described module we have developed a novel method to investigate the scratch resistance of different materials under controlled, reproducible and stable conditions. The scratch module offers a MTP service to set up an experimental procedure in which the needle scratches ten lines linearly separated onto the coated substrate. Each line is supposed to move 80 mm along the negative *x*-position while linearly increasing the force from 5.5 to 20 N on the needle towards the sample, and the scratch speed is set to 1 mm/s.

In this way, a path is obtained and we observe the moment in which the needle breaks through the coating and reaches the metal substrate underneath. For some of our early experiments, it was possible to detect this point both visually and tactilely. Together with the observation that for an individual sample this point is roughly on the same level for the ten scratches, one can turn this method into an automated procedure to quantify the scratch resistance of coatings. Figure 8.6 visualizes the sensor data of a sample as obtained by the OPC UA Server for the needle force in N along the *x*-direction (see figure A.2) on the *y*-axis plotted against the time on the *x*-axis. There are ten different peaks corresponding to each of the scratches. For comparison the same sample is provided as optical image. Each peak roughly follows the same pattern with a controllable amount of fluctuation. Filtering out an individual peak trail, i.e., one individual scratch leads to figure 8.7 (left). The needle begins the scratch on the coating and gradually increases the force (blue bar). The point at which the needle penetrates the coating and reaches the substrate is marked by a green vertical line which stands out as the moment in which the force in the *x*-direction has a sudden drop. Afterwards, the signal has much stronger amplitudes until the needle has arrived the end point (red bar).

[6] https://www.elektrophysik.com/en/products/coating-thickness-measurement-gauges/, accessed on 02.05.2026

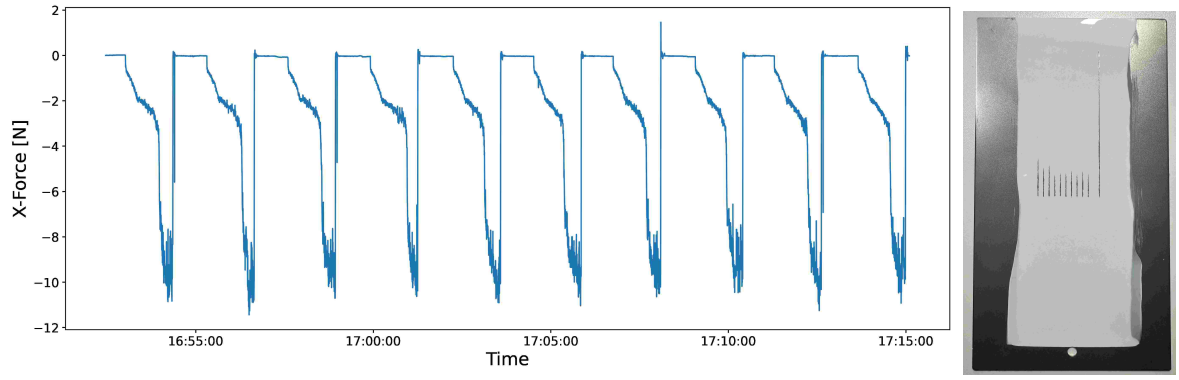


Figure 8.6: Left: Needle force in x-direction (y-axis) against the time (x-axis) for one particular sample. Right: optical image of the sample.

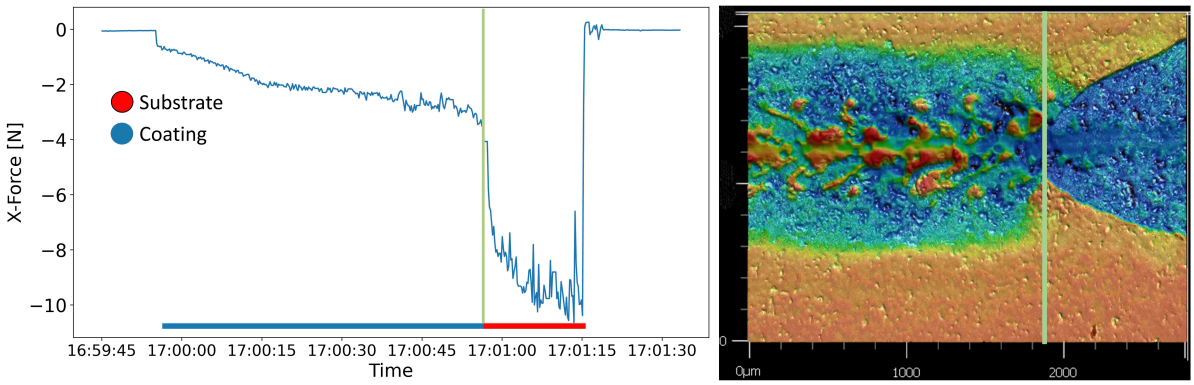


Figure 8.7: Left: Needle force in x-direction (y-axis) against the x-position (x-axis) for one particular sample and scratch. Right: Visualization of the scratch obtained from a laser scanning microscope. The green bar indicates the point of transitioning from the coated surface to the underlying substrate.

The first sudden change in the frictional force is considered as the materials' cohesive failure. The second abrupt change in the data is the adhesive failure[Islam et al., 2022]. Additionally, we evaluated whether this is truly the point at which the breakthrough happens under a laser scanning microscope that allows to probe the height profile of surfaces. These investigations have confirmed this assumption (see figure 8.7, right).

We have collected data of 398 individual scratches from the training samples described in section 8.4.2. All datasets, codes and training details for this study have been exported from SEDAR and made available online. From these, 255 penetrate the coating and 143 do not, due to the coating being too strong or too thick. Each data point is the result of a series of preparation steps which identify the peaks and extract relevant data based on a condition for the 'x-position'. We extract values of 'x-force', find 519 rows to be a maximum length and pad each shorter sample by replicating the values at the boundaries. Hence, there is no time or x-position; instead, each data point is labeled with **the index at which the penetration occurs**. The dataset is split into two separate tasks: (1) classification based on the sensor data of an individual scratch, determine whether there is a breakthrough and (2) regression if there is indeed a breakthrough, find the point at which it occurs. The test set is drawn from

twelve samples applied with a clear coat and scratched ten times each, leading to 120 data points. The needle was unable to penetrate the coating 27 times, the remaining 93 compose the dataset for the regression.

8.5 Evaluating SEDAR's automated machine learning capabilities

This section is based on [Hoseini et al., 2025b], which is an extended version of [Hoseini et al., 2024c].

In chapter 6, we describe SEDAR's data analytics capabilities. In this section, based on the previously introduced datasets, we present evaluation of the three approaches toward automated machine learning:

1. **the AutoMLWrapper** (section 6.1.2)
2. **LLM-assisted code generation for ML** (section 6.1.2)
3. **KG-enhanced data science agents** (section 6.2.4)

The first dataset [Hoseini et al., 2022] contains images of four different types of surface defects (see section 8.2. The images were collected via Laser-scanning microscopy to obtain height profiles on the surfaces and produce an artificial RGB image by a threefold repetition of the height data. This data is evaluated for **object detection**. While the authors dealt with Few-Shot Object Detection [Chen et al., 2021], these solutions are not included in the AutoML libraries, and except for *GPT*4, none of the LLMs had information about such models. Hence, we compare to the given *RetinaNet* baseline. For the same dataset [Hoseini et al., 2022] of surface defects optical images are available for **image classification**. Here we limit the number of available training samples per class to simulate a Few-Shot setting.

The next dataset section 8.3 consists of images of scratch tests on surfaces to be evaluated for **image segmentation** (see section 8.3). The area where the so-called cross-cut test has removed the surface material. A binary mask with the marked regions is available for each image. They use several image augmentations to increase the amount of training data and make the model more robust. For this dataset, only the original images are available, without augmentations.

The last dataset comes from section 8.4, where methods for analyzing the resistance of coatings and paints are discussed again. The breakthrough point of a needle on a coating is identified by using sensor data about the movement and the force exerted by the needle on the surface. Using this data, the authors conduct **tabular regression** to find the position of the penetration through the surface (see section 8.4).

All datasets, code, and training details for this study have been exported from SEDAR and made available online. The experiments are performed in the described *Jupyter* environment equipped with an Nvidia Quadro RTX8000 graphics card, two Intel Xeon Gold 6230R processors, and 754 GB RAM. Training for all models is initially limited to one hour. For the *AutoKeras* library, for which no time limit can be defined, a division into 50 models for 50 epochs is used.

```
%%ai chatgpt Generate a Python code snippet for an Image Segmentation pipeline using the libraries Tensorflow or Pytorch and other
appropriate libraries, optimized for GPU usage. The code should handle a dataframe called gitter_trainv1 with columns 'image' and
'label'. 'image' contains the path to the image file and 'label' contains the path to the mask file. The dataset contains 348 images.
All images are 256px in height, 256px in width, in GRAY mode, and saved as png. Focus on a simple yet effective neural network architecture
suitable for beginners. You may use a pre-trained model and fine-tune it on the dataset. Include necessary imports, dataset loading and
preprocessing steps, model architecture, and training loop. The code should be ready to run in a Python interpreter without modifications.
datagen = ImageDataGenerator(rescale=1./255)
image_generator = datagen.flow_from_dataframe(dataframe=gitter_trainv1, x_col='image', y_col='label',
                                              target_size=(256, 256), color_mode='grayscale',
                                              class_mode='input', batch_size=32)
base_model = MobileNetV2(input_shape=(256, 256, 1), include_top=False)

x = base_model.output
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer=Adam(), loss='binary_crossentropy')
model.fit(image_generator, epochs=10)
```

Figure 8.8: Example prompt using *JupyterAI* with *GPT*4's response to generate the necessary code for solving the image segmentation task. Shortened for brevity.

Setup for LLM-assisted code generation Because working with LLMs can be freely designed by the user, rules to improve reproducibility are defined. LLMs in *GPT4All* are designed for the English language, hence the conversations are conducted in English. Problems are described by the user. We assume that the user cannot provide an exact definition of the sequence of experiments due to a lack of experience. Hence, ideally, the user does not use any ML terminology when working with LLMs. For example, it should not be mentioned that the class distribution of the data set is unbalanced. Instead, it should be checked whether the contextual information provided by the SEDAR AutoML solution is sufficient for the LLM to act accordingly. Any further steps identified by the LLM through continued querying for a more robust ML pipeline are tracked as further effort. The two open-source LLMs that are available in *JupyterAI* through *GPT4All* *nous-hermes-llama2-13b* (Hermes) and *mistral-7b-openorca* (Orca) are used by default, as two representatives with varying parameter sizes and instruction and chat model respectively. Of these, we include the best-performing model in the evaluation. The experiment begins with a single prompt (Figure 8.8) to solve the problem in one shot. If this prompt does not create a valid ML pipeline, e.g., due to errors in the created code, the prompt is modified, for example, to split the problem into several prompts. In this case, it is tracked to what extent additional effort is required from the user and how demanding the changes are. If no approach with open-source LLMs leads to success, *GPT*4 is used.

Criteria

A series of evaluation criteria are defined to conduct a structured evaluation. If, for example, the effort is evaluated, more effort corresponds to a less favorable value, which is why an ascending

color coding is selected (■■■■■). If the completeness of a solution is being evaluated, the process is reversed; higher values are better (■■■■■).

The performance evaluates the predictive capabilities of the models created by the AutoML solutions. Various metrics are considered, e.g. *Mean Average Precision* (MAP) for object detection, and *Intersection over Union* (IuO) for image segmentation. Where possible, the values are determined using the built-in functions of the AutoML library. If the appropriate metric is not implemented, this is highlighted negatively and a corresponding implementation is imported.

For efficiency, the time and resources required to experiment cover the process from data selection to the finished model. The time calculation in turn includes a) the preparations for the experiment (t_{prepare}), such as the dialogs, b) the time spent in the source code (t_{code}), for example by adapting function parameters, and c) the computing time required (t_{compute}). The complexity of the configuration of an experiment or the effort required for any changes to the code is measured by the following color-coded scale as *work*: (1) immediate execution, (2) simple, (3) medium, (4) extensive configuration and (5) individualized configuration with external resources.

The amount of prior knowledge and programming skills required for the configuration and changes to the code for a functioning pipeline is rated.

Finally, the initial model created by the AutoML process is assessed. A good starting situation either directly provides a versatile selection of models and parameters that the user can continue to use, or, especially when working with LLMs, provides sufficient hints to improve the designed pipeline.

8.5.1 Results of AutoMLWrapper and LLM-based Pipelines

The results of the experiments (Tables 8.1 to 8.3) show that the AutoMLWrapper can considerably simplify creating ML pipelines. All results are from a single run. The user only has to make minimal entries in the frontend and can still create powerful models, which is particularly advantageous for users without much experience. The AutoMLWrapper can autonomously train and evaluate models in a short time if the appropriate settings are made. The performances in terms of evaluation metrics are competitive compared to the given baselines for both the AutoMLWrapper and the LLMs. Especially for the classification of paint defects, the training times were remarkably short, which made it possible to quickly create a selection of prototypes. Although the expected user-friendliness is a strength, it also results in reduced flexibility. The user has less control over specific model parameters and the optimization of these. In addition, the performance of the AutoMLWrapper is directly dependent on the capabilities of the underlying libraries. In summary, the module offers an efficient and user-friendly way to quickly perform experiments that also result in powerful models.

More effort corresponds to a less favorable value, thus ascending color coding (■■■■■).
For completeness of a solution the process is reversed; higher values are better (■■■■■).

	$\text{Wrapper}_{\text{gluon}}$	LLM_{gpt4}	Hoseini et al. [2022]
MAP	0.46	0.4	0.458
t_{prepare}(min.)	2	1	
t_{code}(min.)	< 1	10	
t_{compute}(min.)	60	12	
work	■□□□□	■■■□□	
Prior knowledge	■□□□□	■■□□□	
Programming expertise	■□□□□	■■■□□	
Initial model	■■■■□	■■■■■	

Table 8.1: Overview of the experiments related to object detection. Only *AutoGluon* can perform object detection. The open-source models were unable to deal with this task, hence *GPT*4 had to be involved. Distribution of train/test sets: 140/36

	$\text{Wrapper}_{\text{gluon}}$	LLM_{Orca}	Zhang et al. [2022]
IoU	0.82	0.91	0.96
t_{prepare}(min.)	3	1	
t_{code}(min.)	3	15	
t_{compute}(min.)	9	20	
work	■■□□□	■■■■□	
Prior knowledge	■■■□□	■■■□□	
Programming expertise	■□□□□	■■■□□	
Initial model	■■■□□	■■■□□	

Table 8.2: Overview of the experiments in the segmentation data set. Only *AutoGluon* can perform image segmentation. Distribution of train/val/test sets: 174/35/8

	$\text{Wrapper}_{\text{gluon}}$	$\text{Wrapper}^{\text{few-shot}}_{\text{gluon}}$	$\text{Wrapper}_{\text{keras}}$	$\text{LLM}_{\text{Hermes}}$
F1	0.62	0.87	0.6	0.5
F1	0.85	0.99	0.6	0.55
t_{prepare}(min.)	2	2	2	2
t_{code}(min.)	< 1	<1	15	10
t_{compute}(min.)	< 1	< 1	13	17
work	■□□□□	■□□□□	■■□□□	■■■■□
Prior knowledge	■□□□□	■□□□□	■■□□□	■■■■□
Programming expertise	■□□□□	■□□□□	■■□□□	■■□□□
Initial model	■■■■□	■□□□□	■■■□□	■■■■□

Table 8.3: Overview of the image classification experiments with limited training data. *AutoGluon*'s few-shot setting leads to the best results. Distribution of train/test sets: first row 20/79 and second row 80/79

8.5.2 Results of the KG-enhanced Approach

In Table 8.4, we present an evaluation of the selected dataset for tabular regression using three approaches: the AutoMLWrapper, LLM-orchestrated ML code generation, and our proposed knowledge-graph–enhanced method (KGE). Since OpenML is primarily designed for tabular data and contains only limited image-related datasets, this regression task represents the sole case study we report for the KGE approach. The corresponding output is illustrated in Figure 8.9, which shows the report generated by the AssistML backend when extended with the MLSea knowledge graph. The report lists several candidate models, ordered by decreasing relevance. Notably, the top-ranked recommendation exhibits poor predictive performance, whereas the second-ranked recommendation, an AutoSklearn model, achieves near-perfect alignment with the test data, outperforming even the baseline established in Hoseini et al. [2022]. The superior regression performance of the configuration recommended by AssistML can be primarily attributed to the explicit configuration which departs from the AutoMLWrapper by dramatically increasing the memory limit (102 GB vs. the 4 GB default) and replacing cross-validation with a holdout strategy, thereby enabling more complex regressors to be evaluated within the time budget. Together with constraining the ensemble to the top 50 models, these adjustments substantially broaden the effective search space and account for the observed reduction in regression error. However, due to this configuration, the execution time is much longer.

	$\text{Wrapper}_{\text{gluon}}$	$\text{Wrapper}_{\text{keras}}$	$\text{Wrapper}_{\text{sklearn}}$	LLM_{Orca}	Hoseini et al. [2024e]	KGE_1	KGE_2
Mean-Absolute-Error	8.86	64.68	7.38	53.39	4.58	15.67	0.69
t_{prepare}(min.)	2	2	2	2		1	1
t_{code}(min.)	<1	<1	<1	5		< 1	< 1
t_{compute}(min.)	<1	7	60	18		< 1	25
$work_{\text{code}}$	■□□□□	■□□□□	■□□□□	■■□□□		■□□□□	■□□□□
Prior knowledge	■□□□□	■□□□□	■□□□□	■□□□□		■□□□□	■□□□□
Programming expertise	■□□□□	■□□□□	■□□□□	■□□□□		■□□□□	■□□□□
Initial model	■■■■□	■■■□□	■■■■■	■■■■□		■□□□□	■□□□□

Table 8.4: Overview of the regression experiments (1 & 2) on tabular data. Distribution of train/val/test sets: 204/10/41

8.5.3 Discussion

Applying LLMs for code generation and process understanding in ML represents an innovative and promising approach. The open-source models offer the advantages of free availability, easy access to experimentation without direct costs, and special hardware. However, the open-source models' generated answers were often incomplete or lacked precision for a direct application. Many times the code had errors, was not executable. This is also true for older proprietary models such as GPT3.5 (see this notebook for a comparison between various open-source models and GPT3.5, 4o & 5). For the AutoSklearn model as recommened by the KGE method (see figure 8.9), GPT3.5 ommited important model configurations. Especially for more complex queries for model creation and optimization, the answers were cut off prematurely or were so erroneous that considerable effort was required from the user. The results indicated that the models have difficulty understanding and adapting to specific requirements and contexts. This led to generic or inappropriate code, especially for specialized tasks such as image segmentation. In contrast, *GPT*4 provided a significantly improved experience, generating more precise and relevant code that was better adapted to the problem. The answers were almost complete and pointed to additional aspects that could be used directly as an initial model for further questions to improve the model. For example, the open-source models were unable to utilize the COCO format of the object detection dataset, which was no problem for *GPT*4.

8.6 End-To-End ML with LLMs and Semantic Data Management

The following is mainly drawn from our publication [Hoseini et al., 2025a].

This section addresses the efficient and accurate semantic description of the context of an ML project to improve the code generation quality of LLMs for ML. The main problem tackled in this work is the gap between the complex, often unstructured description of input datasets and the ability of LLMs to efficiently process this information to generate code for data wrangling as

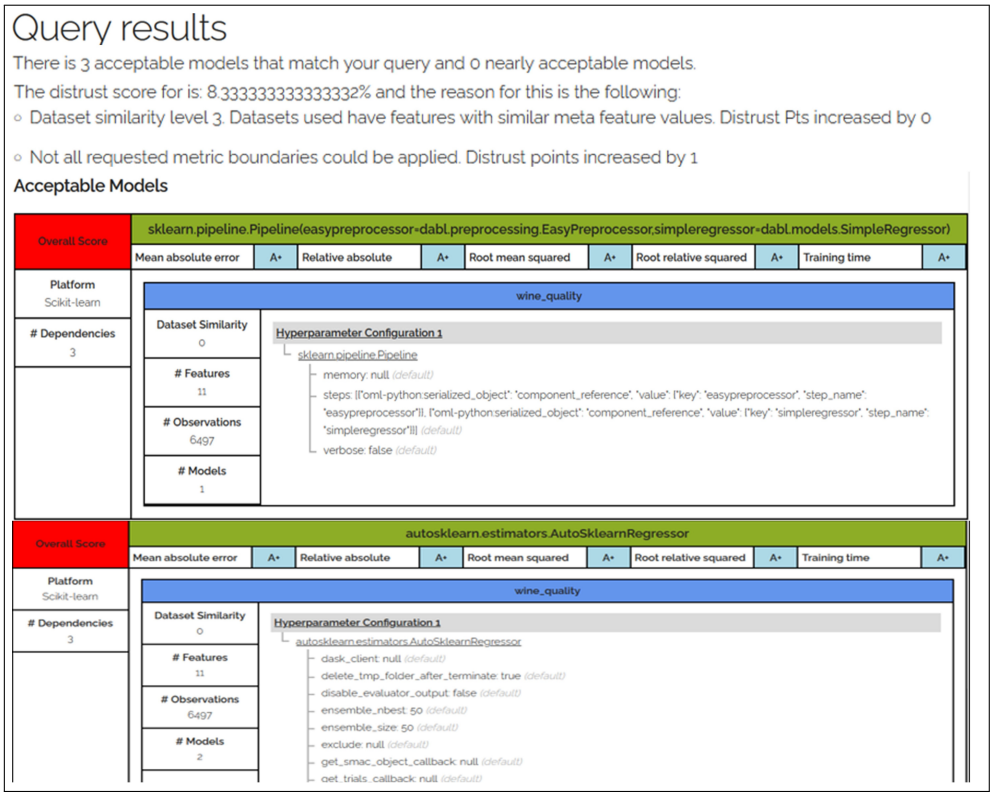


Figure 8.9: A report with two model recommendations generated by AssistML [Zacarias et al., 2023] for the tabular regression dataset.

prerequisite for ML model generation. The proposed solution involves a framework for creating a semantic model that maps data sources to ontologies and encapsulates the relevant context. The focus is the following research question: *How do different contemporary LLMs compare in their ability to generate functionally correct code for data wrangling and ML modeling, when guided by a semantic model describing the input data sources?*

In this proof-of-concept study, we explored the use of semantic data management to support LLM-driven automation of end-to-end ML pipelines in a real-world Chemistry 4.0 setting. A semantic model is used to capture domain knowledge and metadata in a machine-readable form, guiding LLMs through natural language prompts to generate complete data wrangling and ML modeling code. We evaluate several state-of-the-art LLMs on their ability to autonomously produce functionally correct Python code for preprocessing and Gaussian Process modeling. The results show that, when guided by structured semantic context, larger LLMs can reliably generate accurate pipelines, significantly reducing the need for manual intervention. These findings provide an encouraging starting point for further exploration toward leveraging the semantic model to improve the robustness of code generation by systematically integrating relevant information into the generation process, rather than relying solely on the raw intelligence of the LLM.

As an open innovation hub, HIT supports applied industry research, including a recent project aimed at understanding and improving the properties of a legacy lacquer from a re-

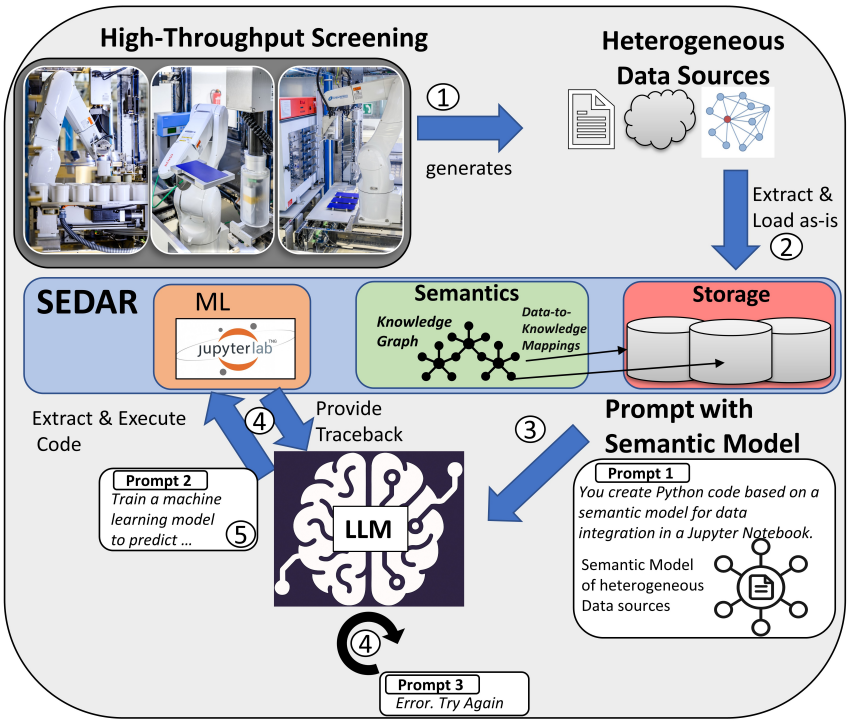


Figure 8.10: End-to-end system combining semantic modeling and LLM prompting for automated code generation

gional company through a design of experiment (DoE) [Karan et al., 2024]. Modeling the DoE required data integration from various sources: Excel sheets reporting individual HTF runs, as well as numerous CSV files containing measurements for characterizing the samples with respect to certain chemical properties. Similar challenges arise across other scientific and industrial domains where experimental data is complex, heterogeneous, and sparsely documented—making automation of ML pipelines highly context-dependent [Gundersen and Kjensmo, 2018, Döhmen et al., 2022].

Related Work Data-driven chemistry leverages experimental data and computational techniques to accelerate the discovery and optimization of chemical processes and materials. The fusion of materials science and modern digital technologies, including robotics, and artificial intelligence for replacing traditional manual, analog and human-intensive work in chemistry [Pyzer-Knapp et al., 2022, Zhang et al., 2023a, Butakova et al., 2021, Burger et al., 2020] is an on-going topic of research. In [Hoseini et al., 2024e] we present an integrated system composed of an automation platform to perform HTF, a characterization module to evaluate the material, and an ML model to accelerate the development of new coatings. The model is trained on data managed by SEDAR. The paper showcases the interplay between chemistry, automation, and data science as a basis for smart manufacturing.

Zhang et al. [2024a] explore the application of HTF and explainable ML techniques for optimizing lacquer formulations. By integrating Gaussian process modeling, Bayesian optimization, and a novel sampling algorithm, the study demonstrates how data-driven approaches can en-

hance coating property predictions, multi-objective optimization, and chemical understanding, ultimately enabling more efficient and rational coating design.

Data Preprocessing with LLMs has recently shown potential [Jaimovitch-López et al., 2023, Santos et al., 2025]. Data integration and wrangling have long been a key focus of the data management community, because they are crucial yet time-consuming steps in data science [Narayan et al., 2022]. One way to involve LLMs to reduce analysts' overhead is to have them generate code for the various data wrangling tasks [Akella and Narayanam, 2025, Li and Döhmen, 2024, Huang et al., 2024], which is in part also the method used in this paper. Others specifically focus on core tasks, such as Column Type Annotation for tabular data [Bodensohn et al., 2024]. Liu et al. [2024a] introduces *AutoDW*, an end-to-end solution for automatic data wrangling that leverages LLMs to transform raw data into more usable formats. The method automates the entire process, including data cleaning & enrichment, feature type inference, and code generation following a template and performs well on selected benchmark datasets, but is limited to a single data table and cannot integrate multiple datasets. Santos et al. [2025] introduces *Harmonia*, a system that leverages LLMs to facilitate the integration of datasets from diverse sources. Further, scaling and automating data preprocessing with LLMs for real-world use cases poses challenges. For example, manual prompt engineering, which is expensive and hard to operationalize as well as fine-tuning of LLMs, which is high in compute and storage costs [Zhang et al., 2024b].

Code generation for ML with LLMs has gained increasing attention in recent years. In a previous work, we compared classical AutoML approaches with LLMs [Hoseini et al., 2024c] showing competitive performance. While AutoML can considerably simplify creating ML pipelines, i.e., autonomously train and evaluate models in a short time, it also comes with reduced flexibility, i.e., the user has less control over specific model parameters and their optimization. Furthermore, most classical AutoML frameworks focus solely on model generation and evaluation, but cannot assist in the preceding stages, i.e., data preprocessing. On the other hand, directly prompting the LLM for code generation as well as process understanding in ML can assist individual data science tasks [Hollmann et al., 2024, Li and Döhmen, 2024], still they lack the ability to perform holistic end-to-end workflows. Both directions struggle to move beyond being merely used as an initial prototype that require manual improvement.

The core challenge lies in orchestrating the entire pipeline, largely because each step depends on the others. For instance, data cleaning and feature engineering are essential precursors, yet they must be adapted to each specific scenario. To bridge this gap various proposals for *End-To-End Data Science Agents* have been proposed (see section 6.2).

Semantic Data Management deals with serializing the context from which heterogeneous data sources arise with Semantic Web techniques (see section 5.1). Figure 5.2, together with Listing 8.1, illustrates a representative excerpt of the semantic model and how it links to the underlying data sources. The semantic model acts as a projection, mapping the entities and relationships from the KG onto the data sources. This model serves as an intermediary layer, bridging the data layer with the knowledge layer. Raw data sets with varying formats are positioned at the lower data layer. The semantic model is represented as a graph comprising nodes

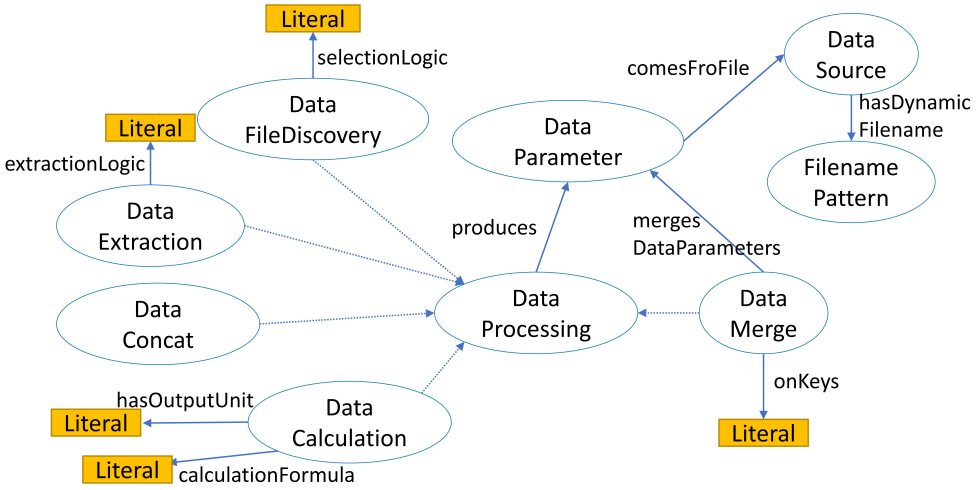


Figure 8.11: Visualization of the ontology underlying the semantic model

```
@prefix mlp: <http://www.semanticweb.org/semanticModel_MLPipeline#> .
mlp:Select_CharacterizationFiles a mlp:DataFileDiscovery;
    mlp:uses mlp:MeasurementNumber, mlp:RUN_ID ;
    mlp:selectionLogic """
    1. Extract Measurementnumber.
    2. Construct directory: 'measurements/Product
       A28-R{RUN_ID}-E{Measurementnumber}-T0/""";
    mlp:produces mlp:xrite, mlp:hazeglossssss, mlp:minitest .
mlp:hazegloss.csv a mlp:DataSource;
    mlp:hasDynamicFilename [mlp:filenameContains "minitest"];
    mlp:hascolumns "gloss20, gloss60, gloss80, gloss85, haze"
    mlp:hasDelimiter "," ;
    rdfs:comment "Haze and gloss measurements file found
       dynamically." .
mlp:Extraction_gloss_85 a mlp:DataExtraction;
    mlp:extractionLogic "Extract gloss85 column from 2nd row of
       hazegloss" ;
    mlp:produces mlp:gloss_85 ;
    mlp:uses mlp:hazegloss .
```

Listing 8.1: Example Triples from the Semantic Model

and edges borrowed from an ontology to denote the mapping and augmented semantics of the data sources. By defining the semantic type and relationships between source attributes in the graph, the semantic model offers a precise description of the data source's intended meaning. So-called semantic labels directly annotate data source elements (e.g., schema attributes) with elements of the semantic model, defined by appropriate predicates and objects.

8.6.1 Context-Aware Code Generation with LLMs and Ontology-Based Semantics

To address the challenges of integrating heterogeneous experimental data for ML in Chemistry 4.0, we introduce a novel approach that combines semantic data modeling with LLMs for automated code generation. As illustrated in Figure 8.10, high-throughput formulation (HTF) generates large volumes of diverse data from automated laboratory workflows. These datasets

Prompt 1: Data Wrangling

You create Python code based on a semantic model for data integration in a Jupyter Notebook.
Semantic model: <complete_semantic_model_as_text>
<example_data_points>
Generate one single and complete code that extracts, processes and merges data from files based on the semantic model to display FinalTable. For this, strictly iterate over run_id first, then over experiment_id in two nested loops.

Prompt 2: ML

Use this table to train a machine learning model to predict viscosity (viskos1) and gloss (gloss60) based on chemical ingredients and process settings in a single model. Do all the necessary preprocessing. Try numerous configurations of Gaussian Process and show how the best model performs on MSE, MAPE & MAE in the scaled space. Generate one complete code.

Prompt 3: Error Handling

<traceback> Look closely at the semantic model and the example data points again. Generate the complete fixed code again.

Figure 8.12: Prompts for End-To-End ML

are ingested and stored in SEDAR, where semantic mappings are created to link raw data to a domain-specific knowledge graph.

Our approach introduces an intermediate semantic layer that enriches raw datasets with structured metadata through ontology-based data-to-knowledge mappings (figure 5.2). This enables context-aware prompting of LLMs to generate Python code for data wrangling and model training within a JupyterLab environment. Semantic prompts (e.g., Prompt 1) guide the LLM by explicitly embedding knowledge about data provenance, structure, and meaning. The generated code is executed and validated within the same environment, allowing users to iteratively refine prompts (Prompt 2) or recover from errors (Prompt 3).

The implementation is divided in two steps, first describing the HTF system in all its details semantically, and refining the obtained model for prompting the LLMs to solve the data wrangling and ML model crafting tasks.

Semantic Description of the Production System In order to describe the HTF equipment and to capture the expert knowledge on the facility in a durable, machine-readable form, an ontology was first developed in *Protégé* [Noy et al., 2001]. The ontology served to identify and define the relevant process steps, data flows, and measurement points involved in the statistical experimental design workflow. By use of the ontology, the system's structures and relationships were refined and serialized in Turtle format using *PLASMA* [Paulus et al., 2023]. The objective was to create a hierarchical representation of all modules, subprocesses, and generated data, thereby ensuring that the core domain of expertise – often confined to spreadsheets and specialist documentation – would become permanently available to other stakeholders without requiring direct consultation of the original domain experts.

Initially, this semantic model integrated data primarily from two sources: (1) an Excel-based report exported from the proprietary software of the HTF equipment containing detailed parameters for each production step, and (2) documentation compiled by automation specialists and employees at the HIT, outlining the responsibilities and interactions of various modules. By incorporating both the physical processes (e.g., substrate transportation from a storage unit to a liquid dosing module) and the measurement data (e.g., dispensed quantities, parameter deviations), the model established explicit links between diverse data points. In doing so, it enabled a clear mapping of input and output variables to specific modules and activities, supporting downstream process control and analysis.

Additional characterization datasets were required to achieve a complete view of the achieved lacquer quality in different experiments of the production process. These datasets included measurements of rheology, thickness, gloss, haze, and spectral measurements, each provided as set of separate CSV files spread across various directories and files. Those are to be found dynamically based on details such as *run*, *experiment* and *measurement IDs*. Expert knowledge was again crucial: in several cases, only a fraction of the recorded measurements were relevant to the analysis (e.g., a single time interval in a longer measurement, or one of multiple measurement points). Domain specialists supplied technical scripts and guidelines indicating which readings to incorporate to avoid the inclusion of invalid data such as uncovered substrate regions.

To integrate these new datasets, the TTL-based *PLASMA* model was reimported into *Protégé*, where custom annotations and data type properties were added. These annotations documented precisely which measurement points, parameters, and conditions were valid and how they mapped to each module's activities. By expanding the model step by step – linking the rheology or gloss, haze data to their respective process nodes – this approach created a richer, fully annotated semantic model.

Iterative refinement With this semantic description, we tried querying different LLMs to see whether they can accurately generate the correct data pipeline. Because in the previous stage, we did not have the LLM's interpretation of the semantic model in mind, the results were sobering.

Initially, the LLM was given a highly detailed prompt mimicking the underlying pipeline logic. While this yielded correct code, it also demanded extensive expertise, undermining the goal of accommodating non-technical users. To mitigate this, the semantic model should contain all necessary information with the assumption that the LLMs would be capable of deriving the core pipeline details directly from the model. In practice, the resulting code deviated from the desired functionality, revealing that the LLMs could answer specific, in-depth queries on the semantic model but struggled to apply the domain knowledge broadly in more complex tasks. These problems arose because many details were missing, misleading or confusing, such as different concepts with very similar namings (e.g., experiment ID vs experiment number), respecting the German local format (i.e.,"," as decimal separator), varying delimiters for CSV files, skipping first rows for specific files, etc. Such information is still considered as describing the data

sources semantically in the sense of SDM, as we strictly did not incorporate any Python-related details into the semantic model. Those are for the LLM to be derived autonomously. Further, the original semantic model had roughly 24k characters across 523 lines. Due to the limited context length of various LLMs we quickly observed that the model is too verbose and hence we shortened it heavily by removing many irrelevant details. This includes unused prefixes and redundant metadata entries. By prompting LLMs for the complete pipeline and carefully eliminating all potential confusion manually, we refined the semantic description iteratively and obtained an accurate semantic description of the data landscape.

This iterative refinement resulted in a semantic model with which anyone or any system interfacing with the model could retrieve the complete set of domain-relevant data without repeatedly consulting the original automation experts. It further ensures that automated code generation via LLMs can accurately leverage the expert knowledge encapsulated within the semantic model.

Figure 8.11 displays a visualization of the semantic model. In order to properly model the complicated data situation, the graph contains classes like `DataProcessing`, `DataParameter`, and connects them to `DataMerge`, `DataExtraction` and `DataFileDiscovery`. Listing 8.1 shows how these classes are actually formed into triples to model data discovery and extraction patterns. The entity `Select_CharacterizationFiles` represents a file selection process that uses `DataParameters` such as `MeasurementNumber` and `RUN_ID` to construct a directory path and locate specific CSV files.

8.6.2 Evaluation

The goal is to establish a user-friendly pathway for generating code to build training tables and a corresponding ML by relying on simple natural language prompts rather than specialized programming knowledge. All programming code is executed on a *JupyterHub* connected to the SEDAR system. The target functionality centers on the existing scripts: listing F.1 and listing F.2.

Code generation evaluation loop with retry-based error correction We exposed only the enriched model to the LLMs retaining abstract prompts wherever possible. This strategy sought to balance minimal user input with sufficient domain detail. For this, we composed concise prompts, one for the data harmonization step (Prompt 1), and one for crafting the ML (Prompt 2).

We observed that all LLMs struggled with grasping the necessary intricacies to understand the pipeline fully. For this reason, a strategic hint was necessary without which none of the experiments succeeded. The hint explicitly mentions a nested loop over the `run_id` first, then over the `experiment_id` (see listing F.1). Possibly, there are other strategies to solve the problem, and one could have let the LLM find out by itself, however we observed that without the hint none of the generated codes was functional.

We allow iterations of error handling. Once the code is generated, it will be executed in

	GPT4o-mini	GPT4o	GPTo1	Gemini2Pro	Grok3	Llama3.1	DeepSeekR1
Release	May'24	May'24	Sep'24	Mar'25	Feb'25	Jul'24	Jan'25
# Parameters	Est. 8–20B	Est. 400–500B	?	1.56Trillion	?	405B	671B
Success	XXX	✓✓✓	X✓✓	✓✓✓	✓✓X	XXX	✓✓X
Av. total time	13:32min	02:34min	04:54min	00:49min	01:35min	04:20min	05:54min
Av. time p. iteration	03:23min	01:57min	01:52min	00:49min	00:28min	01:15min	2:58min
Av. retries	3	0.5	2	0	2.3	3	1.6

Table 8.5: Evaluating the data wrangling task for various LLMs

	Baseline	GPT4o	GPTo1	Gemini2Pro	Grok3
MAE	0.0062	0.0059	0.086	0.015	0.0056
MSE	0.044	0.042	0.056	0.058	0.0416
MAPE	0.15	0.16	0.14	0.29	0.15

Table 8.6: Evaluating the ML task for various LLMs

SEDAR's notebook environment. If the code produces an error or the generated table does not equal the target table, if exists, the error message alongside the prompt 3 are directed back to the LLM to give it the chance of fixing the code. In *Jupyter*, the verbosity is set to $\%xmode\ Plain$ to ensure a simplified and concise traceback. If the returned table has invalid values, the error prompt includes this info. This process is allowed up to three times, after which we consider the experiment as failed and we track how many iterations were needed in each run. Due to probabilistic nature of the LLMs, we observed that each individual run within a session returns different answers for the exact same prompt. For this reason, we repeat the experiment for each LLM three times and report the averages in Table 8.5.

Results For each tested LLM, we first report functional success, which requires functional correctness, i.e.,the code executes without runtime errors and produces the exact target dataset. This also requires pipeline completeness (file discovery, wrangling) and accuracy of the semantic alignment. Further, we report resilience via retry success rate by the average number of iterations required for correct output. As can be seen, the prompt complexity is kept at a minimum, assuming to have a suitable semantic model at hand, anyone can execute the procedure with minimal human edits. If, and only if, the correct table has been created, we move on to crafting the ML model with Prompt 2. Minimal instructions with regards to the target columns and the particular ML method are necessary. Here, we report the performance against regression metrics (MAE, MSE, MAPE) as compared to the human baseline. We published the datasets, the semantic model, the experimental settings, the evaluation details, and the LLM conversations on GitHub.

Table 8.5 shows the results of the wrangling task. We have tested a series of openly available and commercial LLMs. The open-source models have been tested via *DeepInfra*[7], all

[7] https://deepinfra.com/, accessed on 02.05.2026

others through their respective commercial website.

Our tests quickly revealed that smaller models fail at the task. This includes LLama3.3:70b, Gemini2.5Flash, Qwen2.5-Coder-32B-Instruct, DeepSeek-R1-Turbo and others. As a representative of a most powerful small model we include GPT4o-mini here, but omit the others. In contrast, the largest models, such as GPT4o and Gemini2Pro seem to have no problem with the complexity of the task. Here, Gemini is much faster than GPT4o in our tests. We observed that small models have difficulty to recall all details of the semantic model. The initial prompt counted with *OpenAIs* GPT4o tokenizer has 4508 tokens. If errors occur, retries increase the conversation length and recalling specific details from the semantic models becomes more challenging and erroneous. This is known as the *Lost-in-the-Middle* problem. As Liu et al. [2024b] show, larger models exhibit superior capability of parsing lengthy sequences as compared to small models. Partly due to positional encoding limitations, transformer models struggle to retrieve key facts (like how to parse a specific data value) at various positions within long prompts. Adding more info into a long prompt will be ineffective, because position matters. Our experiments confirm that strategic info is best placed near the start or end.

Regarding the open-source models, LLama3.1, although a rather large model, was not capable of solving the task once. To our surprise, DeepSeekR1 was able to solve the task on two occasions with only two error corrections. GPTo1 and Grok were able to solve the task with multiple retries of error corrections twice, but failed to perform the complete series.

The ML task is much less challenging to solve, and even the smaller models succeed without problems (see Table 8.6). To make the results comparable we standardized the use of the MinMaxScaler instead of StandardScaler. The generated models have performances competitive to the human baseline. This comes partly from the fact, that the task itself has minor complexity and can be solved with basic ML knowledge. This confirms that LLMs are capable of automating ML prototyping.

These results highlight that while data wrangling remains a challenging and context-sensitive task even for state-of-the-art LLMs, the automation of basic ML modeling is already within reach using minimal prompts—provided the data is well-structured and semantically described.

8.6.3 Discussion

This research demonstrates that LLMs are capable of generating complex and functionally correct code for data wrangling when provided with a semantic model. However, this success is heavily reliant on the sheer capacity of the LLM to interpret and reason over large amounts of input text. In our setup, the semantic model was passed as part of the prompt in textual form, causing the LLM to process it as extended natural language rather than leveraging its inherent graph structure. As a result, the highly curated, interconnected, and structured nature of the semantic model—typically a strength in domain-specific contexts—remained largely unexploited.

This points to a broader limitation in current LLM capabilities: they struggle to effectively utilize structured knowledge representations, especially when presented in unstructured for-

mats. In addition to the Lost-in-the-Middle problem, LLMs lack persistent access to external knowledge sources [Chang et al., 2024]. To address this, recent research has proposed *KG-enhanced LLMs*, which integrate knowledge graphs into the LLMs to enable more grounded, consistent, and context-aware generation [Hu et al., 2023].

In our view, bridging this gap requires a deeper integration of structured knowledge into the LLM pipeline. Techniques such as RAG, when combined with knowledge graphs, offer a promising direction [Hogan et al., 2025]. While RAG provides dynamic access to relevant information, the knowledge graph ensures that this information is structured and semantically grounded. Therefore, future work will focus on crafting a method for unifying RAG, knowledge graphs, and LLM-based data science agents into a coherent architecture—where RAG supports broad retrieval, knowledge graphs provide structured domain context, and LLMs perform generative reasoning—thus enabling more robust and semantically informed automation of data workflows.

One idea is to enhance the error-based retry loop through a retrieval component that dynamically feeds the LLM with the relevant facts from the semantic model based on the traceback. Beyond that, future enhancements may include a hybrid agent architecture, where specialized agents collaborate—one for parsing the semantic model, another for generating code, and a third for validating the output. This modular design, inspired by multi-agent LLM systems, distributes responsibilities and reduces the cognitive load on a single model. Another promising direction is to implement an interactive debugging agent, allowing the LLM to engage in a fallback dialog mode when errors persist. In this mode, the model can ask clarifying questions about ambiguities in the semantic model or dataset structure (e.g., "Is `ExperimentNumber` the same as `ExperimentID`?") before generating a revised solution. These additions could significantly improve robustness, interpretability, and user collaboration in future LLM-based data science workflows.

8.7 SEDAR's Research Contributions and Evaluation with DATER

The research objectives of SEDAR, initially introduced in chapter 4, can now be revisited in light of the complete end-to-end architecture presented in this chapter. While chapter 4 focused on the core architectural layers, the following contributions reflect their realization and extension across the full data lifecycle, including analytics, automated machine learning, and intelligent user interaction.

Lifecycle-integrated semantic interpretation of data: As introduced in Chapter 4, SEDAR integrates semantic interpretation and metadata management directly into the data lifecycle and elevates metadata from passive documentation to an operational coordination layer. With the full architecture now revealed, this principle is shown to govern not only ingestion, versioning, semantic labeling, and lineage, but also data discovery, analytical execution, automated machine learning, and agent-based workflow orchestration. Metadata thus functions as an executable control plane that enables semantic interoperability, lineage-aware analytics, and

AI-assisted automation across heterogeneous data sources and tools.

A unified and storage-agnostic data access abstraction: Building on the unified logical data access model introduced in chapter 4, the completed SEDAR architecture demonstrates how strict decoupling of logical data access from physical storage and execution engines enables scalable processing, OBDA-based querying, and uniform analytical workflows. This abstraction allows heterogeneous storage backends to be combined or replaced without affecting ingestion pipelines, metadata management, or user-facing interaction layers, and proves essential for supporting automated ML pipelines and cross-domain analytics without enforcing schema-on-write constraints.

Semantic interoperability as a foundation for analytics and automation: While semantic interoperability was initially framed as a prerequisite for data integration, the full system demonstrates how knowledge graphs, ontologies, and semantic labels act as first-class enablers for analytics, automated machine learning, and intelligent agents. Semantic context is systematically exploited to guide feature selection, workflow composition, and tool invocation, thereby reducing manual effort and increasing robustness in complex, multistep data science workflows.

Architecture-level integration of AI-assisted interaction: Beyond the core architecture presented in Chapter 4, SEDAR introduces a systematic integration of AI-assisted interaction mechanisms, including natural-language interfaces and multi-agent coordination. These components expose architectural semantics to non-technical users while remaining grounded in formally defined metadata, schemas, and ontologies. This demonstrates how intelligent interaction can be embedded at the architectural level rather than added as an isolated interface layer.

Taken together, these contributions confirm that SEDAR constitutes an architectural synthesis spanning ingestion, semantic data management, analytics, and interaction.

Evaluation with DATER

In chapter 3, we evaluated a selection of real-world systems using the DATER framework. Now that SEDAR has been fully described, we apply the same evaluation procedure to it.

From a control and governance perspective (D1–D3), SEDAR initially exhibits centralized characteristics. Security and policy enforcement remain basic. In terms of data modeling and understanding (D4–D6), SEDAR supports heterogeneous formats, semantic enrichment, and the use of ontologies and knowledge graphs as first-class citizens, making it highly suitable for knowledge-driven applications. Metadata management is automated and partially AI-augmented. Data processing (D7-D9) in SEDAR treats integration as a foundational concern, leverages Apache Spark for scalable processing, supports OBDA, and implements the full ML lifecycle following MLOps principles.

SEDAR partially overlaps with data lakehouse and warehouse paradigms due to its sup-

port for relational processing, SQL-based querying, and ML workflows. However, it does not enforce schema-on-write DW1 and the same storage type and data format across sources DLH1 and does not prioritize OLAP-centric optimization DW7. Given its semantic foundation, integration-centric design, and AI-enhanced capabilities, SEDAR increasingly fulfills key **data fabric** criteria (DF2-DF4), although currently SEDAR implements a centralized storage layer DF1. Hence, it moves away from the scope of classical data lakes, warehouses, and lakehouses by offering semantic interoperability, self-service capabilities and advanced automation across the data lifecycle.

8.8 Conclusion

This chapter addressed the research questions (see section 1.2) by demonstrating how the SEDAR platform performs in real-world, data-driven chemistry scenarios and how its architecture aligns with the DATER framework.

The three applied ML studie show how SEDAR meets the domain-specific data-management requirements of industrial chemistry, thereby providing empirical evidence for RQ2. These datasets also enabled a systematic evaluation of the AutoMLWrapper and the KG-enhanced Data Science Agent (section 8.5), addressing RQ3 and RQ4 by demonstrating the usefulness of standardized automated ML pipelines and LLM-driven workflow support.

The semantic end-to-end ML workflow (section 8.6) further illustrated how ontology-based metadata, and context-aware code generation can automate complex analytical processes, confirming the practical value of the semantic extensions developed in response to RQ3.

Finally, the evaluation of SEDAR using the DATER framework (section 8.7) provides the applied validation of RQ1, showing how the system aligns with and satisfies the technical requirements of modern data architectures when assessed through a structured, requirement-driven framework.

Together, these applications demonstrate that SEDAR is not only architecturally well-founded but also operationally effective in industrial practice—capable of integrating heterogeneous data, automating ML workflows, and making advanced analytics accessible to domain experts in chemistry.

9

Conclusion and Outlook

اسرار ازل را نه تو دانی و نه من
وین حرف معما نه تو خوانی و نه من
هست از پس پرده گفت‌وگوی من و تو
چون پرده بر افتد، نه تو مانی و نه من

The secrets of eternity neither you know, nor I.
This riddle's solution neither you read, nor I.
Behind the veil is the talk of you and me,
Since when the veil falls neither you remain, nor I.

Omar Khayyam

Despite the many insights gained throughout this dissertation, they represent only a small contribution to a dynamically expanding field. Even the most rigorous inquiry cannot unveil all the mysteries of our world. What we have achieved here is no absolute truth, but a structured attempt to understand a complex domain more clearly, while knowing that every answer leads to new questions, and that some veils may never fully lift.

9.1 Key contributions

To address the challenge of effective management of data within the context of the data-driven chemistry within the i²DACH research project, we identified and addressed four research questions (see section 1.2): We summarize the key contributions and describe how they address the respective research questions:

We addressed RQ1 in chapter 3 by presenting DATER, a framework for evaluating contemporary data architectures based on technical requirements: We conducted a comparative analysis of modern data architectures, including data warehouses, data lakes, semantic data lakes, data lakehouses, data fabrics, and data meshes. In doing so, we clarified overlapping terminology and provided a standardized vocabulary for describing data architectures within their environments and for characterizing the components that compose them. Finally, we evaluated

real-world systems using the DATER framework and nine dimensions, mapping each to one of the architecture types.

RQ2 is addressed in chapters 4 to 6 and 8. First, we proposed SEDAR and described its components. We then showed how SEDAR implements semantic data management techniques. Building on this, we developed extensions for the system to integrate and document automated machine learning techniques and discussed future approaches for combining data science agents with knowledge graphs to address domain-specific requirements. Finally, we presented data-driven use cases from the HIT-Institute in which SEDAR was deployed for industrial chemistry.

RQ3 is addressed in section 2.9 and chapter 5. In the background section, we defined terms, provided definitions, and structured the field along semantic labels, semantic models, and ontology-based integration and access. We then explored how SEDAR integrates LLMs with knowledge graphs to automate and enhance semantic type detection and relation inference. Furthermore, we integrated the PLASMA system into SEDAR to enable GUI-supported creation and refinement of semantic models across multiple datasets. Finally, we described how this approach enables semantic querying across heterogeneous data sources based on the semantic models created with PLASMA.

RQ4 was addressed in chapters 7 and 8: We presented a natural language interface for SEDAR to address challenges of inaccessibility. In addition, we introduced the AutoMLWrapper and a method based on LLMs to increase ML pipeline automation and democratize access to advanced analytics. Finally, we proposed an approach grounded in semantic data management to provide accurate descriptions of the context of an ML project, thereby improving the code generation quality of LLMs for machine learning.

9.2 Discussion, Limitations and Directions for Future Work

While SEDAR presents a novel and multifaceted approach to semantic data management, it remains a research prototype with several practical limitations that impact its readiness for broader adoption. Despite the architectural innovations and integration of state-of-the-art technologies (e.g., ontology-based data access, AutoML pipelines, and LLM-driven semantics), SEDAR, in its current form, exhibits a number of unresolved technical issues and substantial refactoring would be necessary to ensure maintainability, scalability, and performance in production environments. In view of the many mature proprietary solutions like Databricks or Azure, the system as it stands is unlikely to see wide-scale adoption. It should be viewed primarily as a proof of concept rather than as a deployable product. Nonetheless, the current prototype provides a valuable experimental platform for evaluating novel ideas in semantic enrichment, multi-agent systems, and data automation workflows that has been, is right now and will still be deployed for various use cases. Looking ahead, Vincent Hermann, a master's

student and research assistant in our lab has expressed interest in continuing development of SEDAR and is currently working on consolidating the code base and setting up a more straightforward method for installation. This transition in responsibility may offer an opportunity to revisit core design decisions, focus on improving code quality as well as several further research directions:

- Progression of SEDAR even further toward a data fabric. The storage layer, which at the moment is deeply attached and integrated in to the code base, could be externalized and organized more flexibly to contribute to decentralization.
- With the rise of LLMs, the current potential for AI-based self-services for data management, such as the NLI, is immense. Here, semantics-aware data science agents are particularly exciting and a natural next step for which initial investigation exists already.
- Connectors for the IDSA data space initiative could be valuable to share data from and to SEDAR via the Gaia-X ecosystem [Solmaz et al., 2022].

The integration of a graph-based multi-agent system for natural language interaction with SEDAR represents a forward-looking step toward more intelligent, flexible, and context-aware user interfaces. Agent-based approaches are particularly well-suited to complex data environments, offering modularity, reasoning capabilities, and a clear path toward adaptive behaviors. From this perspective, it is reasonable to argue that agents represent a promising direction for the future of human–data interaction. However, the current implementation of the natural language interface remains tightly coupled to SEDAR's architecture. As such, questions remain regarding its adaptability, portability, and scalability in heterogeneous settings. Comparative studies across multiple domains and data platforms will be critical to demonstrate the generalizability of the agents in different operational contexts. A next step could apply the idea in the context of industrial chemistry to answer queries and execute workflows in the context of the HTF equipment at the HIT.

One of the central pillars of SEDAR is its support for semantic data management through ontology-driven modeling and ontology-based data access. While these technologies offer significant advantages in terms of data integration, interoperability, and machine-readable semantics, they also introduce substantial complexity, especially from a user experience perspective. Creating accurate and meaningful semantic models is a non-trivial task that requires deep understanding of ontologies, data structures, and domain semantics. Even with support of LLMs and tools like PLASMA, the process of mapping source data to ontological structures remains cognitively demanding and time-intensive. The knowledge required to perform this task is often beyond the reach of non-technical users or even many domain experts without formal training in semantic technologies. Further complicating the usability landscape is the use of SPARQL for data querying via OBDA. Although SPARQL is powerful and expressive, it has a steep learning curve and lacks the intuitiveness expected by users accustomed to more conventional interfaces (e.g., SQL, dashboards, or natural language queries). As a result, OBDA in its current form is inaccessible to most non-technical stakeholders, limiting the practical utility of semantic systems in collaborative, interdisciplinary settings. Addressing these barriers will require the development of more user-friendly interfaces, potentially mediated by LLMs or guided work-

flows, that abstract away the low-level complexity of semantic modeling and query formulation.

The rapid advancement of LLMs has begun to reshape the landscape of semantic technologies. While LLMs are powerful in pattern recognition and linguistic reasoning, they lack explicit, structured, and verifiable semantic representations. In this domain knowledge graphs remain essential as they enable logic-based reasoning, data validation, and structured querying. Rather than seeing LLMs and KGs as competing paradigms, it is more accurate to view them as complementary. LLMs can help bootstrap, enrich, and interact with KG, while KGs can provide ground truth, structure, and semantic anchors for LLM-driven applications. As explained in previous section of this PhD thesis, future work should aim to tightly integrate these technologies.

Bibliography

Ahmed Abbasi, Suprateek Sarker, and Roger HL Chiang. Big data research in information systems: Toward an inclusive research agenda. *Journal of the association for information systems*, 17(2):3, 2016.

Nora Abdelmageed and Sirko Schindler. Jentab meets semtab 2021's new challenges. In *Proc. SemTab*, volume 3103 of *CEUR WS*, pages 42–53, 2021. URL http://ceur-ws.org/Vol-3103/paper4.pdf.

Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. *Data profiling*, volume 10. Springer, 2019.

Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

Amani Abu Rumman and Lina Al-Abbadi. Structural equation modeling for impact of data fabric framework on business decision-making and risk management. *Cogent Business & Management*, 10(2):2215060, 2023.

Russell L Ackoff. From data to wisdom. *Journal of applied systems analysis*, 16(1):3–9, 1989.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

Panagiotis Aivaliotis, Konstantinos Georgoulias, and George Chryssolouris. The use of digital twin for predictive maintenance in manufacturing. *International Journal of Computer Integrated Manufacturing*, 32(11):1067–1080, 2019.

Ashlesha Akella and Krishnasuri Narayanam. Data wrangling task automation using code-generating language models. *arXiv preprint arXiv:2502.15732*, 2025.

Montasser AKERMI, Mohamed Ali HADJ TAIEB, and Mohamed BEN AOUICHA. Data virtualization enabling distributed data architectures: Data fabric and data mesh. *International Journal of Computer Information Systems and Industrial Management Applications*, 15:12–12, 2023.

Majid Al-Ruithe, Elhadj Benkhelifa, and Khawar Hameed. A systematic literature review of data governance and cloud data governance. *Pers. Ubiquitous Comput.*, 23(5-6):839–859, 2019. doi: 10.1007/S00779-017-1104-3. URL https://doi.org/10.1007/s00779-017-1104-3.

Ahmed Ali. Integration of an advanced user interface for semantic data management into a semantic data lake. Master's thesis, RWTH Aachen University, Aachen, Germany, October 2023. Supervised by Prof. Dr. Stefan Decker, second supervisor: Prof. Dr. Christoph Quix, advisor: Sayed Hoseini, M.Sc.

Sridhar Alla and Suman Kalyan Adari. *What Is MLOps?* Apress, Berkeley, CA, 2021. ISBN 978-1-4842-6549-9.

Bradley P Allen, Lise Stork, and Paul Groth. Knowledge engineering using large language models. *arXiv preprint arXiv:2310.00637*, 2023.

Micah M Alvord, Fengyu Lu, Boyang Du, and Chia-An Chen. Big data fabric architecture: How big data and data management frameworks converge to bring a new generation of competitive advantage for enterprises. *Enterprise Architecture Professional Journal*, 2022.

Qi An, Chihua Ying, Yuqing Zhu, Yihao Xu, Manwei Zhang, and Jianmin Wang. Ledd: Large language model-empowered data discovery in data lakes. *arXiv preprint arXiv:2502.15182*, 2025.

Carlos Buil Aranda, Olivier Corby, Souripriya Das, Lee Feigenbaum, Paula Gearon, Birte Glimm, Steve Harris, Sandro Hawke, Ivan Herman, Nicholas Humfrey, Nico Michaelis, Chimezie Ogbuji, Matthew Perry, Alexandre Passant, Axel Polleres, Eric Prud'hommeaux, Andy Seaborne, and Gregory Todd Williams. SPARQL 1.1 query language for RDF. W3C Recommendation, 2013. URL https://www.w3.org/TR/sparql11-query/. Accessed on 2024-10-18.

Inês Araújo Machado, Carlos Costa, and Maribel Yasmina Santos. Advancing data architectures with data mesh implementations. In *International Conference on Advanced Information Systems Engineering*, pages 10–18. Springer, 2022.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. Delta lake: high-performance acid table storage over cloud object stores. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, 2020.

Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*, volume 8, page 28, 2021.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433*, 2023.

Peb R Aryan, Fajar J Ekaputra, Elmar Kiesling, A Min Tjoa, and Kabul Kurniawan. Rmlx: Mapping interface for integrating open data with linked data exploration environment. In *Proc. ICICoS*, pages 113–118. IEEE, 2017.

Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Comput. Surv.*, 54(5), may 2021. ISSN 0360-0300.

Abdulrahman Ashraf, Ahmed Hassan, and Hani Mahdi. Key lessons from microservices for data mesh adoption. In *2023 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, pages 1–8, 2023. doi: 10.1109/MIUCC58832.2023.10278300.

Pouya Ataei and Alan Litchfield. The state of big data reference architectures: A systematic literature review. *IEEE Access*, 10:113789–113807, 2022.

Maurizio Atzori, Angelo Ciaramella, Claudia Diamantini, BD Martino, Salvatore Distefano, Tullio Facchinetti, Fabrizio Montecchiani, Antonino Nocera, Giancarlo Ruffo, Roberto Trasarti, et al. Dataspaces: Concepts, architectures and initiatives. In *CEUR workshop proceedings*, volume 3606. CEUR-WS, 2024.

Roberto Avogadro and Marco Cremaschi. Mantistable v: A novel and efficient approach to semantic table interpretation. In *SemTab@ ISWC*, pages 79–91, 2021.

Sarah Azzabi, Zakiya Alfughi, and Abdelkader Ouda. Data lakes: A survey of concepts and architectures. *Computers*, 13(7):183, 2024.

Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

Deepika Badampudi, Claes Wohlin, and Kai Petersen. Experiences from using snowballing and database searches in systematic literature studies. In *Proceedings of the 19th international conference on evaluation and assessment in software engineering*, pages 1–10, 2015.

Ada Bagozi, Devis Bianchini, Valeria De Antonellis, Massimiliano Garda, and Michele Melchiori. Personalised exploration graphs on semantic data lakes. In *Proc. OTM Conf.*, volume 11877 of *LNCS*, pages 22–39. Springer, 2019. doi: 10.1007/978-3-030-33246-4_2.

Jie Bao, Diego Calvanese, Bernardo Cuenca Grau, Martin Dzbor, Achille Fokoue, Christine Golbreich, Sandro Hawke, Ivan Herman, Rinke Hoekstra, Ian Horrocks, Elisa Kendall, Markus Krötzsch, Carsten Lutz, Deborah L. McGuinness, Boris Motik, Jeff Pan, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, Alan Ruttenberg, Uli Sattler, Michael Schneider, Mike Smith, Evan Wallace, Zhe Wu, Antoine Zimmermann, Jeremy Carroll, Jim Hendler,

and Vipul Kashyap. Owl 2 web ontology language. W3C Recommendation, 2012. URL https://www.w3.org/TR/owl2-overview/. Accessed on 2024-05-23.

David A. C. Beck, James M. Carothers, Venkat R. Subramanian, and Jim Pfaendtner. Data science: Accelerating innovation and discovery in chemical engineering. *AIChE Journal*, 62 (5), 2016. doi: http://dx.doi.org/10.1002/aic.15192.

David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 turtle. *World Wide Web Consortium*, pages 18–31, 2014.

Leon Beer. Enhancement of a data lake system for MLOps. Master's thesis, Hochschule Niederrhein, Krefeld, Germany, January 2023. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Amin Beheshti, Boualem Benatallah, Reza Nouri, and Alireza Tabebordbar. Corekg: a knowledge lake service. *Proc. VLDB Endow.*, 11(12):1942–1945, 2018. doi: 10.14778/3229863.3236230. URL http://www.vldb.org/pvldb/vol11/p1942-beheshti.pdf.

T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. RFC 3986, Internet Engineering Task Force (IETF), January 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

Tim Berners-Lee. Semantic web on xml - the semantic web stack, 2000. URL https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html. Accessed on 15.10.2024.

Tim Berners-Lee, James Hendler, and Ora Lassila. A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific american*, 284(5): 34–43, 2001.

Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, 36(21):12809–12844, 2024.

Devis Bianchini, Valeria De Antonellis, and Massimiliano Garda. A semantics-enabled approach for personalised data lake exploration. *Knowledge and Information Systems*, 66(2): 1469–1502, 2024.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data. In *Semantic Services, Interoperability and Web Applications - Emerging Concepts*, pages 205–227. CRC Press, 2011a. doi: 10.4018/978-1-60960-593-3.ch008.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011b.

Ivo Blohm, Felix Wortmann, Christine Legner, and Felix Köbler. Data products, data mesh, and data fabric: New paradigm (s) for data and analytics? *Business & Information Systems Engineering*, pages 1–10, 2024.

Jan Bode, Niklas Kühl, Dominik Kreuzberger, and Carsten Holtmann. Towards avoiding the data mess: Industry insights from data mesh implementations. *IEEE Access*, 2024.

Jan-Micha Bodensohn, Ulf Brackmann, Liane Vogel, Matthias Urban, Anupam Sanghi, and Carsten Binnig. LLMs for data engineering on enterprise data. *Proceedings of the VLDB Endowment. ISSN*, 2150:8097, 2024.

Ludwig Boltzmann. Über die beziehung zwischen dem zweiten hauptsatze der mechanischen wärmetheorie und der wahrscheinlichkeitsrechnung respektive den sätzen über das wärmegleichgewicht. *Sitzungsberichte der Kaiserlichen Akademie der Wissenschaften in Wien, Mathematisch-Naturwissenschaftliche Klasse*, 76:373–435, 1877.

Wolfgang Bongardt, Martin Lorek, and Ekkehard Wilde. Method for the recognition and evaluation of defects in reflective surface coatings, March 23 1999. US Patent 5,887,077.

Ranjit Bose. Advanced analytics: opportunities and challenges. *Industrial Management & Data Systems*, 109(2):155–172, 2009.

Will Brackenbury, Rui Liu, Mainack Mondal, Aaron J. Elmore, Blase Ur, Kyle Chard, and Michael J. Franklin. Draining the data swamp: A similarity-based approach. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358279. doi: 10.1145/3209900.3209911.

Stefan Braun, Manuela Dalibor, Nico Jansen, Matthias Jarke, István Koren, Christoph Quix, Bernhard Rumpe, Manuel Wimmer, and Andreas Wortmann. Engineering digital twins and digital shadows as key enablers for industry 4.0. In *Digital transformation: core technologies and emerging topics from a computer science perspective*, pages 3–31. Springer, 2023.

Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0. W3C Recommendation, 1998. URL https://www.w3.org/TR/1998/REC-xml-19980210.html.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

BSI Standards. BS EN ISO 1518-1: 2011—Paints and varnishes—determination of scratch resistance part 1—constant-loading method. 2011.

S. J. Bull and E. G. Berasetegui. An overview of the potential of quantitative coating adhesion measurement by scratch testing. *Tribology International*, 39(2):99–114, February 2006. ISSN 0301-679X. doi: 10.1016/j.triboint.2005.04.013.

Peter Buneman. Semistructured data. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 117–121, 1997.

Andreas Burgdorf, Alexander Paulus, André Pomp, and Tobias Meisen. VC-SLAM - A handcrafted data corpus for the construction of semantic models. *Data*, 7(2):17, 2022. doi: 10.3390/data7020017.

Benjamin Burger, Phillip M. Maffettone, Vladimir V. Gusev, Catherine M. Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M. Alston, Buyi Li, Rob Clowes, Nicola Rankin, Brandon Harris, Reiner Sebastian Sprick, and Andrew I. Cooper. A mobile robotic chemist. *Nature*, 583(7815):237–241, July 2020. doi: 10.1038/s41586-020-2442-2.

Frederik Burgsmüller. Development of a software library for few-shot learning based on the crosstransformer model. Master's thesis, Hochschule Niederrhein, Krefeld, Germany, January 2022. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Adrian Burton and Andrew Treloar. Designing for discovery and re-use: the 'ands data sharing verbs' approach to service decomposition. *International Journal of Digital Curation*, 4(3): 44–56, 2009.

Maria A. Butakova, Andrey V. Chernov, Oleg O. Kartashov, and Alexander V. Soldatov. Data-Centric Architecture for Self-Driving Laboratories with Autonomous Discovery of New Nanomaterials. *Nanomaterials*, 12(1):12, December 2021. ISSN 2079-4991. doi: 10.3390/nano12010012.

Vijay Kumar Butte and Sujata Butte. Enterprise data strategy: a decentralized data mesh approach. In *2022 International Conference on Data Analytics for Business and Industry (ICDABI)*, pages 62–66. IEEE, 2022.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Ontology-based data access and integration. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*. Springer, 2 edition, 2018. doi: 10.1007/978-1-4614-8265-9_80667. URL https://doi.org/10.1007/978-1-4614-8265-9_80667.

Diego Calvanese, Linfang Ding, Alessandro Mosca, and Guohui Xiao. Realizing ontology-based reusable interfaces for data access via virtual knowledge graphs. In *Proc. Italian SIGCHI (CHItaly)*, pages 35:1–35:5. ACM, 2021. doi: 10.1145/3464385.3464744.

Marçal Mora Cantallops, Salvador Sánchez-Alonso, and Elena García-Barriocanal. A systematic literature review on Wikidata. *Data Technol. Appl.*, 53(3):250–268, 2019. doi: 10.1108/DTA-12-2018-0110.

Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015. URL http://sites.computer.org/debull/A15dec/p28.pdf.

Gavin Carothers and Andy Seaborne. RDF 1.1 n-triples. W3C Working Group Note, 2014. URL https://www.w3.org/TR/n-triples/. Accessed on 2024-05-23.

Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. Data management for machine learning: A survey. *IEEE TKDE*, 35(5):4646–4667, 2023.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, jan 2024. ISSN 2157-6904. doi: 10.1145/3641289. Just Accepted.

Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

Angelica Chen, David M. Dohan, and David R. So. Evoprompting: Language models for code-level neural architecture search, 2023a.

Chuer Chen, Yuqi Liu, Danqing Shi, Shixiong Cao, and Nan Cao. Datascout: Automatic data fact retrieval for statement augmentation with an LLM-based agent. *arXiv preprint arXiv:2504.17334*, 2025.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.137. URL https://aclanthology.org/2024.findings-acl.137/.

Tung-I Chen, Yueh-Cheng Liu, Hung-Ting Su, Yu-Cheng Chang, Yu-Hsiang Lin, Jia-Fong Yeh, and Winston H. Hsu. Dual-awareness attention for few-shot object detection. *CoRR*, abs/2102.12152, 2021.

Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. Symphony: Towards natural language query answering over multi-modal data lakes. In *CIDR*, pages 1–7, 2023b.

Alexandros Chortaras and Giorgos Stamou. D2RML: integrating heterogeneous data and web services into custom RDF graphs. In *Proc. LDOW Workshop*, volume 2073 of *CEUR WS*, 2018. URL http://ceur-ws.org/Vol-2073/article-07.pdf.

Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

Andrea Colaço, Ahmed Kirmani, Gregory A Howland, John C Howell, and Vivek K Goyal. Compressive depth map acquisition using a single photon-counting detector: Parametric signal processing meets sparsity. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 96–102. IEEE, 2012.

Óscar Corcho, Freddy Priyatna, and David Chaves-Fraga. Towards a new generation of ontology based data access. *Semantic Web*, 11(1):153–160, 2020. doi: 10.3233/SW-190384.

Justin Cunningham. Netflix data mesh: Composable data processing - justin cunningham. Online video, 2020. Available at: https://www.youtube.com/watch?v=TO_IiN06jJ4 [Accessed 06.02.2025].

Olivier Curé, Robin Hecht, Chan Le Duc, and Myriam Lamolle. Data integration over NoSQL stores using access path based mappings. In *Proc. DEXA*, volume 6860 of *LNCS*, pages 481–495. Springer, 2011. doi: 10.1007/978-3-642-23088-2_36.

Olivier Curé, Fadhela Kerdjoudj, David Faye, Chan Le Duc, and Myriam Lamolle. On the potential integration of an ontology-based data access approach in NoSQL stores. *Int. J. Distributed Syst. Technol.*, 4(3):17–30, 2013. doi: 10.4018/jdst.2013070102.

Prithiviraj Damodaran. FlashRank, Lightest and Fastest 2nd Stage Reranker for search pipelines., December 2023. URL https://github.com/PrithivirajDamodaran/FlashRank.

Benjamin Danker. Detection of defects on paint surfaces using machine learning. Bachelor's thesis, Hochschule Niederrhein, Krefeld, Germany, September 2021. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Recommendation, September 2012. URL https://www.w3.org/TR/r2rml/.

Ioannis Dasoulas, Duo Yang, and Anastasia Dimou. MLSEA: a semantic layer for discoverable machine learning. In *European Semantic Web Conference*, pages 178–198. Springer, 2024.

Christopher John Date. *An introduction to database systems*. Pearson Education India, 1977.

Thomas H Davenport et al. Putting the enterprise into the enterprise system. *Harvard business review*, 76(4):121–131, 1998.

Ali Davoudian, Liu Chen, and Mengchi Liu. A survey on NoSQL stores. *ACM Computing Surveys (CSUR)*, 51(2):1–43, 2018.

Michael DeBellis, Livia Pinera, and Christopher Connor. Interoperability frameworks: Data fabric and data mesh architectures. In *Data Science with Semantic Technologies*, pages 267–286. CRC Press, 2023.

Stefan Decker, Sergey Melnik, Frank van Harmelen, Dieter Fensel, Michel C. A. Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Comput.*, 4(5):63–74, 2000. doi: 10.1109/4236.877487. URL https://doi.org/10.1109/4236.877487.

Google DeepMind. Gemini, 2025. URL https://deepmind.google/technologies/gemini/flash/. Accessed: April 1, 2025.

Zhamak Dehghani. How to move beyond a monolithic data lake to a distributed data mesh, May 2019. URL https://martinfowler.com/articles/data-monolith-to-mesh.html.

Zhamak Dehghani. Data mesh principles and logical architecture, 2020. URL https://martinfowler.com/articles/data-mesh-principles.html. Accessed on 18.01.2024.

Zhamak Dehghani. *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, Sebastopol, CA, 2022. ISBN 978-1492092391. URL https://www.oreilly.com/library/view/data-mesh/9781492092346/.

Deloitte GmbH. *Chemie 4.0*. VDI e.V., Frankfurt am Main, 2017.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: table understanding through representation learning. *SIGMOD Rec.*, 51(1):33–40, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: efficient finetuning of quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Claudia Diamantini, Paolo Lo Giudice, Lorenzo Musarella, Domenico Potena, Emanuele Storti, and Domenico Ursino. A new metadata model to uniformly handle heterogeneous data lake sources. In *Proc. ADBIS Short Papers & Workshops*, volume 909 of *CCIS*, pages 165–177. Springer, 2018. doi: 10.1007/978-3-030-00063-9_17.

Henrik Dibowski and Stefan Schmid. Using knowledge graphs to manage a data lake. 2021.

Henrik Dibowski, Stefan Schmid, Yulia Svetashova, Cory Henson, and Tuan Tran. Using semantic technologies to manage a data lake: Data catalog, provenance and access control. In *Proc. Scalable Semantic Web Knowledge Base Systems Workshop*, volume 2757 of *CEUR WS*, pages 65–80, 2020.

Roger Dietrich. *Paint Analysis: 2nd Revised Edition*. Vincentz Network, 2021. ISBN 9783748604358. doi: 10.1515/9783748604358. URL https://doi.org/10.1515/9783748604358.

Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In *Proc. LDOW Workshop*, volume 1184 of *CEUR WS*, 2014. URL http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

James Dixon. Pentaho, hadoop, and data lakes | james dixon's blog, 2010. URL https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. [Online; accessed 22-Oct-2024].

James Dixon. Data lakes revisited, 2014. URL https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/. [Online; accessed 22-Oct-2024].

Carl Doersch, Ankush Gupta, and Andrew Zisserman. Crosstransformers: spatially-aware few-shot transfer. *arXiv preprint arXiv:2007.11498*, 2020.

Anton Dolhopolov, Arnaud Castelltort, and Anne Laurent. Implementing federated governance in data mesh architecture. *Future Internet*, 16(4):115, 2024.

Jugendra Dongre, Gend Lai Prajapati, and SV Tokekar. The role of apriori algorithm for finding the association rules in data mining. In *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pages 657–660. IEEE, 2014.

Erik Dubrov. From data integration to financial insights: Analysing a centralized data platform and its use in Power BI reporting. Bachelor's thesis, Hochschule Niederrhein, Faculty of Engineering and Computer Science, Krefeld, Germany, August 2025. Written in collaboration with SWK AG. Examiner: Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The bigdawg polystore system. *SIGMOD Rec.*, 44(2):11–16, August 2015. ISSN 0163-5808. doi: 10.1145/2814710.2814713. URL https://doi.org/10.1145/2814710.2814713.

Till Döhmen, Madelon Hulsebos, Christian Beecks, and Sebastian Schelter. Gitschemas: A dataset for automating relational data preparation tasks. In *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, pages 74–78, 2022. doi: 10.1109/ICDEW55742.2022.00016.

Pavan Edara and Mosha Pasumansky. Big metadata : When metadata is big data. *Proc. VLDB Endow.*, 14(12):3083–3095, 2021. doi: 10.14778/3476311.3476385. URL http://www.vldb.org/pvldb/vol14/p3083-edara.pdf.

Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In *Proc. SEMANTiCS & SuCCESS*, volume 1695 of *CEUR WS*, 2016.

Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. HANDLE - A generic metadata model for data lakes. In *Proc. DaWaK*, volume 12393 of *LNCS*, pages 73–88. Springer, 2020. doi: 10.1007/978-3-030-59065-9_7.

Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. Modeling metadata in data lakes - A generic model. *Data Knowl. Eng.*, 136: 101931, 2021.

Rebecca Eichler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. From data asset to data product - the role of the data provider in the enterprise data marketplace. In Johanna Barzen, Frank Leymann, and Schahram Dustdar, editors, *Service-Oriented Computing - 16th Symposium and Summer School, SummerSOC 2022, Hersonissos, Crete, Greece, July 3-9, 2022, Revised Selected Papers*, volume 1603 of *Com-*

munications in Computer and Information Science, pages 119–138. Springer, 2022. doi: 10.1007/978-3-031-18304-1_7. URL https://doi.org/10.1007/978-3-031-18304-1_7.

Abel Elekes, Martin Schaeler, and Klemens Boehm. On the various semantics of similarity in word embedding models. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–10, 2017. doi: 10.1109/JCDL.2017.7991568.

Kemele M. Endris, Philipp D. Rohde, Maria-Esther Vidal, and Sören Auer. Ontario: Federated query processing against a semantic data lake. In *Proc. DEXA*, volume 11706 of *LNCS*, pages 379–395. Springer, 2019. doi: 10.1007/978-3-030-27615-7_2_9.

Erichsen GmbH & Co. KG. Cross hatch cutting and adhesion testing: Scratch hardness tester 430 p-iv-smart, 2021. URL https://www.erichsen.de/en-gb/products/surface-testing/adhesion-impact-and-elasticity-testing/cross-hatch-cutting-and-adhesion-testing/cross-hatch-cutting-and-adhesion-testing-6/scratch-hardness-tester-430-p-smart-1?set_language=en-gb. Accessed: September 2021.

EVONIK. Speed circuit for coating formulations shortens development time. Press release, 2017. URL https://corporate.evonik.com/en/speed-circuit-for-coating-formulations-shortens-development-time-101273.html. Accessed on 08.10.2024.

Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *arXiv preprint arXiv:2210.01922*, 2022.

Michael Färber and Achim Rettinger. A statistical comparison of current knowledge bases. In *SEMANTiCS (Posters & Demos)*, pages 18–21, 2015.

Mina Farid, Alexandra Roatis, Ihab F Ilyas, Hella-Franziska Hoffmann, and Xu Chu. Clams: Bringing quality to data lakes. In *Proc. SIGMOD*, 2016.

Naglaa Fathy, Walaa Gad, and Nagwa Badr. A unified access to heterogeneous big data through ontology-based semantic integration. In *Proc. ICICIS*, pages 387–392. IEEE, 2019.

U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, 1996.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.139. URL https://aclanthology.org/2020.findings-emnlp.139/.

F. Louis Floyd. The challenges of using artificial intelligence in coatings development. *CoatingsTech Magazine*, August 2021. URL https://www.paint.org/challenges-of-using-artificial-intelligence-in-coatings-development/. Accessed on 10.10.2024.

B Fotovvati, N Namdari, and A Dehghanghadikolaei. On coating techniques for surface protection: A review. *Journal of manufacturing and material processing*, 2019.

Robert M French. The turing test: the first 50 years. *Trends in cognitive sciences*, 4(3):115–122, 2000.

Johannes Frey, Lars-Peter Meyer, Natanael Arndt, Felix Brei, and Kirill Bulert. Benchmarking the abilities of large language models for RDF knowledge graph creation and comprehension: How well do LLMs speak turtle? *arXiv preprint arXiv:2309.17122*, 2023.

Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W Paton. Data wrangling for big data: Challenges and opportunities. In *19th International Conference on Extending Database Technology*, pages 473–478, 2016.

Giuseppe Futia, Antonio Vetrò, and Juan Carlos De Martin. Semi: A semantic modeling machine to build knowledge graphs with graph neural networks. *SoftwareX*, 12:100516, 2020. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2020.100516.

Sainyam Galhotra and Udayan Khurana. Semantic search over structured data. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, page 3381–3384, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3417426.

Herminio García-González, Iovka Boneva, Slawek Staworko, José Emilio Labra Gayo, and Juan Manuel Cueva Lovelle. Shexml: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Comput. Sci.*, 6:e318, 2020. doi: 10.7717/peerj-cs.318.

Inc. Gartner. Gartner says beware of the data lake fallacy, 2014. URL https://www.gartner.com/en/newsroom/press-releases/2014-07-28-gartner-says-beware-of-the-data-lake-fallacy. [Online; accessed 29-Jun-2025].

Sandra Geisler, Maria-Esther Vidal, Cinzia Cappiello, Bernadette Farias Lóscio, Avigdor Gal, Matthias Jarke, Maurizio Lenzerini, Paolo Missier, Boris Otto, Elda Paja, et al. Knowledge-driven data ecosystems toward data transparency. *ACM Journal of Data and Information Quality (JDIQ)*, 14(1):1–12, 2021.

Debananda Ghosh. *Mastering Microsoft Fabric*. Springer, 2024.

Corinna Giebler, Christoph Gröger, Eva Hoos, Rebecca Eichler, Holger Schwarz, and Bernhard Mitschang. The data lake architecture framework. In *BTW 2021*, pages 351–370. Gesellschaft für Informatik, Bonn, 2021.

Joan Giner-Miguelez, Abel Gómez, and Jordi Cabot. Using large language models to enrich the documentation of datasets for machine learning. *arXiv preprint arXiv:2404.15320*, 2024.

Qdrant Solutions GmbH. Qdrant - high-performance, massive-scale vector database and vector search engine for the next generation of ai. also available in the cloud, 2024. URL https://github.com/qdrant/qdrant. Accessed: October 15, 2024.

Lukasz Golab and M Tamer Ozsu. *Data stream management*. Springer Nature, 2022.

Sergio Alejandro Gómez and Pablo Rubén Fillottrani. Leveraging large language models for ontology-based data access: A preliminary analysis. In *XXX Congreso Argentino de Ciencias de la Computación (CACIC)(La Plata, 7 al 11 de octubre de 2024)*, 2024.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Alex Gorelik. *The Enterprise Big Data Lake*. O'Reilly Media, Sebastopol, CA, USA, 2016.

Hans W Gottinger. *Internet Economics: Models, Mechanisms and Management*. Bentham Science Publishers, 2017.

Simon Gottschalk and Elena Demidova. Tab2kg: Semantic table interpretation with lightweight semantic profiles. *Semantic Web*, 13(3):571–597, 2022.

Todd Groff and Thomas Jones. *Introduction to knowledge management*. Routledge, 2012.

Christoph Gröger. There is no ai without data. *Communications of the ACM*, 64(11):98–108, 2021.

Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, et al. Large language models orchestrating structured reasoning achieve kaggle grandmaster level. *arXiv preprint arXiv:2411.03562*, 2024.

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993. doi: 10.1006/KNAC.1993.1008. URL https://doi.org/10.1006/knac.1993.1008.

Zhenzhen Gu, Francesco Corcoglioniti, Davide Lanti, Alessandro Mosca, Guohui Xiao, Jing Xiong, and Diego Calvanese. A systematic overview of data federation systems, 2022. Submitted, not yet accepted.

Ramanathan V Guha, Dan Brickley, and Steve Macbeth. Schema. org: evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51, 2016.

Odd Erik Gundersen and Sigbjørn Kjensmo. The state of the art: Reproducibility in machine learning. *ACM Computing Surveys (CSUR)*, 51(1):1–36, 2018.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 16813–16848. PMLR, 2024.

Shubham Gupta, Pedro A. Szekely, Craig A. Knoblock, Aman Goel, Mohsen Taheriyan, and Maria Muslea. Karma: A system for mapping structured sources into the semantic web. In *ESWC Satellite Events - Revised Selected Papers*, volume 7540 of *LNCS*, pages 430–434. Springer, 2012. doi: 10.1007/978-3-662-46641-4_40.

Rihan Hai. *Data integration and metadata management in data lakes*. PhD thesis, Dissertation, RWTH Aachen University, 2020, 2020.

Rihan Hai, Sandra Geisler, and Christoph Quix. Constance: An intelligent data lake system. In *Proc. SIGMOD*, 2016.

Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2023. doi: 10.1109/TKDE.2023.3270101.

Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Managing google's data lake: an overview of the goods system. *IEEE Data Eng. Bull.*, 2016. URL http://sites.computer.org/debull/A16sept/p5.pdf.

Armin Haller, Axel Polleres, Daniil Dobriy, Nicolas Ferranti, and Sergio J Rodríguez Méndez. An analysis of links in Wikidata. In *European Semantic Web Conference*, pages 21–38. Springer, 2022.

Ahmed A. Harby and Farhana Zulkernine. From data warehouse to lakehouse: A comparative review. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 389–395, 2022. doi: 10.1109/BigData55660.2022.10020719.

Ahmed A Harby and Farhana Zulkernine. Data lakehouse: A survey and experimental study. *Information Systems*, page 102460, 2024.

Dick Hardt. Rfc 6749: The oauth 2.0 authorization framework, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Shengtao He. Achieving tool calling functionality in LLMs using only prompt engineering without fine-tuning. *arXiv preprint arXiv:2407.04997*, 2024.

YCWE Hechler, Maryela Weihrauch, and Yan Catherine Wu. Data fabric and data mesh approaches with ai. *Berkeley, CA, USA: Apress Berkeley*, 2023.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models. In *International Conference on Artificial Intelligence and Statistics*, pages 5549–5581. PMLR, 2023.

Martin Hentschel, Jonathan Dees, Florian Funke, Max Heimel, and Ismail Oukid. Building a data management system for the cloud: Lessons learned and future directions. *Datenbank-Spektrum*, 25(1):17–28, 2025. doi: 10.1007/S13222-025-00494-9. URL https://doi.org/10.1007/s13222-025-00494-9.

Vincent Herrmann. LLM-basierte Code-Generierung für Machine-Learning-Pipelines mit semantischen Beschreibungen. Bachelor's thesis, Hochschule Niederrhein, Fachbereich Elektrotechnik und Informatik, Krefeld, Germany, October 2024. Prüfer: Prof. Dr. Christoph Quix.

Tony Hey, Stewart Tansley, Kristin Michele Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle. Rmleditor: A graph-based mapping editor for linked data mappings. In *Proc. ESWC*, volume 9678 of *LNCS*, pages 709–723. Springer, 2016. doi: 10.1007/978-3-319-34129-3_43.

Pieter Heyvaert, Anastasia Dimou, Ben De Meester, Tom Seymoens, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, and Erik Mannens. Specification and implementation of mapping rule visualization and editing: Mapvowl and the rmleditor. *J. Web Semant.*, 49: 31–50, 2018a. doi: 10.1016/j.websem.2017.12.003.

Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *ESWC Satellite Events, Revised Selected Papers*, volume 11155 of *LNCS*, pages 213–217. Springer, 2018b. doi: 10.1007/978-3-319-98192-5_40.

Yannik Hildebrandt. Behavior driven development: Analysis of a company. Bachelor's thesis, Hochschule Niederrhein, Krefeld, Germany, February 2021. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Pascal Hitzler. A review of the semantic web field. *Commun. ACM*, 64(2):76–83, 2021. doi: 10.1145/3397512. URL https://doi.org/10.1145/3397512.

Aidan Hogan. Web of data. In *The Web of Data*, pages 15–57. Springer, 2020.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022. doi: 10.1145/3447772.

Aidan Hogan, Xin Luna Dong, Denny Vrandečić, and Gerhard Weikum. Large language models, knowledge graphs and search engines: A crossroads for answering users' questions. *arXiv preprint arXiv:2501.06699*, 2025.

Noah Hollmann et al. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36, 2024.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An LLM agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

Sayed Hoseini. Semantic data management for managing heterogeneous data sources in Chemistry 4.0. In *Doctoral Consortium at ISWC 2024 co-located with 23rd International Semantic Web Conference (ISWC 2024)*, 2024.

Sayed Hoseini, Gaoyuan Zhang, Alexander Jongbloed, Christian Schmitz, and Christoph Quix. Automated defect detection for coatings via height profiles obtained by laser-scanning microscopy. *Machine Learning with Applications*, 10:100413, 2022.

Sayed Hoseini, Ahmed Ali, Haron Shaker, and Christoph Quix. SEDAR: A semantic data reservoir for heterogeneous datasets. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 5056–5060, 2023.

Sayed Hoseini, Andreas Burgdorf, Alexander Paulus, Tobias Meisen, Christoph Quix, and André Pomp. Challenges and opportunities of LLM-augmented semantic model creation for dataspaces. In *European Semantic Web Conference*, pages 183–200. Springer, 2024a.

Sayed Hoseini, Andreas Burgdorf, Alexander Paulus, Tobias Meisen, Christoph Quix, and André Pomp. Towards LLM-augmented creation of semantic models for dataspaces. In *The Second International Workshop on Semantics in Dataspaces, co-located with the Extended Semantic Web Conference*, 2024b.

Sayed Hoseini, Maximilian Ibbels, and Christoph Quix. Enhancing machine learning capabilities in data lakes with AutoML and LLMs. In *European Conference on Advances in Databases and Information Systems*, pages 184–198. Springer, 2024c.

Sayed Hoseini, Johannes Theissen-Lipp, and Christoph Quix. A survey on semantic data management as intersection of ontology-based data access, semantic modeling and data lakes. *Journal of Web Semantics*, 81:100819, 2024d.

Sayed Hoseini, Gaoyuan Zhang, Dominik Polke, Alvin Surjana, Lasse Wagner, Christian Schmitz, and Christoph Quix. Coatings intelligence: Data-driven automation for Chemistry 4.0. In *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 1–6. IEEE, 2024e.

Sayed Hoseini, Vincent Herrmann, and Christoph Quix. End-To-End ML with LLMs and semantic data management: Experiences from Chemistry 4.0. In *Proceedings of the Workshop on Data Management for End-to-End Machine Learning*, DEEM '25, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400719240. doi: 10.1145/3735654.3735942. URL https://doi.org/10.1145/3735654.3735942.

Sayed Hoseini, Maximilian Ibbels, Maximilian Knoll, and Christoph Quix. Towards enhancing data science agents with semantics. *Computer Science and Information Systems*, 2025b. under review.

Sayed Hoseini, Christoph Quix, and Stefan Decker. Conversational data management: A multi-agent natural language interface for semantic data lakes. 2026a. Submitted for publication, under review.

Sayed Hoseini, Christoph Quix, and Stefan Decker. Data Architectures and their Technical Requirements (DATER). 2026b. Submitted for publication, under review.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (MCP): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.

Linmei Hu, Zeyi Liu, Ziwang Zhao, Lei Hou, Liqiang Nie, and Juanzi Li. A survey of knowledge enhanced pre-trained language models. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–19, 2023. doi: 10.1109/TKDE.2023.3310002.

Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. Major technical advancements in apache hive. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1235–1246, 2014.

Junjie Huang, Daya Guo, Chenglong Wang, Jiazhen Gu, Shuai Lu, Jeevana Priya Inala, Cong Yan, Jianfeng Gao, Nan Duan, and Michael R. Lyu. Contextualized data-wrangling code generation in computational notebooks. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 1282–1294, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695503. URL https://doi.org/10.1145/3691620.3695503.

Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International*

Conference on Knowledge Discovery & Data Mining, KDD '19, page 1500–1508, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330993.

Thorsten Händler. Balancing autonomy and alignment: A multi-dimensional taxonomy for autonomous LLM-powered multi-agent architectures, 2023. URL https://arxiv.org/abs/2310.03659.

Maximilian Ibbels. Extension of a data lake system with automated machine learning. Master's thesis, Hochschule Niederrhein, Krefeld, Germany, March 2024. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Kingsley Uyi Idehen. Semantic web layer cake tweak, explained. Technical report, OpenLink Software, 2017. URL https://medium.com/openlink-software-blog/semantic-web-layer-cake-tweak-explained-6ba5c6ac3fab. Accessed on 15.10.2024.

Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. Sdm-rdfizer: An RML interpreter for the efficient creation of RDF knowledge graphs. In *Proc. ACM CIKM*, pages 3039–3046, 2020.

Enrique Iglesias, Samaneh Jozashoori, and Maria-Esther Vidal. Scaling up knowledge graph creation to large and heterogeneous data sources. *Journal of Web Semantics*, 75:100755, 2023. ISSN 1570-8268. doi: https://doi.org/10.1016/j.websem.2022.100755.

William H Inmon. *Building the data warehouse*. John wiley & sons, 2005.

William H Inmon and Daniel Linstedt. *Data architecture: a primer for the data scientist: big data, data warehouse and data vault*. Morgan Kaufmann, 2014.

Dama International. *DAMA-DMBOK: Data Management Body of Knowledge (2nd Edition)*. Technics Publications, LLC, Denville, NJ, USA, 2017. ISBN 1634622340.

International Organization for Standardization. ISO/IEC TR 20547-1:2020: Information technology — Big data reference architecture — Part 1: Framework and application process. https://www.iso.org/standard/71275.html, 2020. Accessed on 02.06.2025.

International Organization for Standardization. Information technology—metadata registries (mdr)—part 1: Framework. Tech. rep. ISO/IEC 11179-1, International Organization for Standardization, Geneva, Switzerland, 2023.

Md Didarul Islam, Himendra Perera, Benjamin Black, Matthew Phillips, Muh-Jang Chen, Greyson Hodges, Allyce Jackman, Yuxuan Liu, Chang-Jin Kim, Mohammed Zikry, et al. Template-free scalable fabrication of linearly periodic microstructures by controlling ribbing defects phenomenon in forward roll coating for multifunctional applications. *Advanced Materials Interfaces*, 9(27):2201237, 2022.

ISO2409:2020. Paints and varnishes, cross-cut test. 2020.

Nils Jahnke, Katrin Bendiek, and Malina Klueß. Data Mesh und Data Spaces – Zwei Konzepte für dezentrale Datenökosysteme. Whitepaper Fraunhofer ISST, https://www.bk.admin.ch/dam/bk/de/dokumente/dti/DatenoekosystemSchweiz/Grundlagen/Fraunhofer-ISST_Data-Mesh-und-Data-Spaces_Whitepaper%20.pdf.download.pdf/Fraunhofer-ISST_Data-Mesh-und-Data-Spaces_Whitepaper%20.pdf, 2023. Accessed on 02.06.2025.

Gonzalo Jaimovitch-López, Cèsar Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, and María José Ramírez-Quintana. Can language models automate data wrangling? *Machine Learning*, 112(6):2053–2082, 2023.

Ramesh Jain and Pinaki Sinha. Content without context is meaningless. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1259–1268, 2010.

Matthias Jarke and Christoph Quix. On warehouses, lakes, and spaces: The changing role of conceptual modeling for data integration. In Jordi Cabot, Cristina Gómez, Oscar Pastor, Maria-Ribera Sancho, and Ernest Teniente, editors, *Conceptual Modeling Perspectives*, pages 231–245. Springer, 2017. doi: 10.1007/978-3-319-67271-7_16. URL https://doi.org/10.1007/978-3-319-67271-7_16.

Matthias Jarke, Manfred A. Jeusfeld, C. Quix, and Panos Vassiliadis. Architecture and Quality in Data Warehouses: An Extended Repository Approach. *Information Systems*, 24(3):229–253, 1999.

Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses*. Springer-Verlag, 2 edition, 2003.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), mar 2023. ISSN 0360-0300. doi: 10.1145/3571730.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Kavitha Srinivas, and Vincenzo Cutrona. Results of semtab 2020. In *CEUR Workshop Proceedings*, volume 2775, pages 1–8, 2020.

Andrew Josey. *ArchiMate® 3.1-A Pocket Guide*. Van Haren, 2019.

Ademar Crotti Junior, Christophe Debruyne, Rob Brennan, and Declan O'Sullivan. Funul: a method to incorporate functions into uplift mapping languages. In *Proc. iiWAS*, pages 267–275. ACM, 2016. doi: 10.1145/3011141.3011152.

Elem Güzel Kalayci, Irlán Grangel-González, Felix Lösch, Guohui Xiao, Anees ul Mehdi, Evgeny Kharlamov, and Diego Calvanese. Semantic integration of bosch manufacturing data using virtual knowledge graphs. In *Proc. ISWC*, volume 12507 of *LNCS*, pages 464–481. Springer, 2020.

Andreas Kallmuenzer, Alexey Mikhaylov, Mihaela Chelaru, and Wojciech Czakon. Adoption and performance outcome of digitalization in small and medium-sized enterprises. *Review of Managerial Science*, pages 1–28, 2024.

Uday Kamath, Kevin Keenan, Garrett Somers, and Sarah Sorenson. Large language models: A deep dive, 2024.

Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare, editors, *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3363–3372. ACM, 2011. doi: 10.1145/1978942.1979444. URL https://doi.org/10.1145/1978942.1979444.

Dogancan Karan, Guoying Chen, Nicholas Jose, Jiaru Bai, Paul McDaid, and Alexei A Lapkin. A machine learning-enabled process optimization of ultra-fast flow chemistry with multiple reaction metrics. *Reaction Chemistry & Engineering*, 9(3):619–629, 2024.

Shubhra Kanti Karmaker et al. AutoML to date and beyond: Challenges and opportunities. *ACM Comput. Surv.*, 54(8), oct 2021. ISSN 0360-0300.

Jens Kaufmann, Sayed Hoseini, Pascal Quindeau, Christoph Quix, and Sylvia Ruschin. Bringing it all together–gemeinschaftlich aktiv lernen am virtuell geteilten bildschirm in der hochschule und digital. *HMD Praxis der Wirtschaftsinformatik*, 58(6):1378, 2021a.

Jens Kaufmann, Stephan Kühnel, René Theuerkauf, Emal M Alekosai, Sayed Hoseini, Uwe Neuhaus, and Michael Schulz. Where is the science in data science projects? 2021b.

Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. Chorus: foundation models for unified data discovery and exploration. *arXiv preprint arXiv:2306.09610*, 2023.

Tomaž Kern, Eva Krhač, Marjan Senegačnik, and Benjamin Urh. Digitalizing the paints and coatings development process. *Processes*, 7(8):539, 2019.

Yasar Khan, Antoine Zimmermann, Alokkumar Jha, Vijay Gadepally, Mathieu d'Aquin, and Ratnesh Sahay. One size does not fit all: querying web polystores. *Ieee Access*, 7:9598–9617, 2019.

Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, Özgür L. Özçep, Mikhail Roshchin, Nina Solomakhina, Ahmet Soylu, Christoforos Svingos, Sebastian Brandt, Martin Giese, Yannis E. Ioannidis, Steffen Lamparter, Ralf Möller, Yannis Kotidis, and Arild Waaler. Semantic access to streaming and static data at siemens. *J. Web Semant.*, 44: 54–74, 2017. doi: 10.1016/j.websem.2017.02.001.

Kim Katharina Kieselbach, Matthias Nöthen, and Henning Heuer. Development of a visual inspection system and the corresponding algorithm for the detection and subsequent classification of paint defects on car bodies in the automotive industry. *Journal of Coatings Technology and Research*, 16(4):1033–1042, 2019.

Ralph Kimball and Joe Caserta. *The data warehouse ETL toolkit*. John Wiley & Sons, 2004.

Ralph Kimball and Margy Ross. *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013.

Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792, 2015. doi: 10.1109/BigData.2015.7364082.

Maximilian Knoll. Entwicklung eines LLM-basierten Data-Science-Agenten zur automatisierten Modellgenerierung. Master's thesis, Hochschule Niederrhein, Fachbereich Elektrotechnik und Informatik, Krefeld, Germany, April 2025. Prüfer: Prof. Dr. rer. nat. Christoph Quix, Zweitprüfer: M. Sc. Sayed Hoseini.

R. Kohavi, N. J. Rothleder, and E. Simoudis. Emerging trends in business analytics. *Communications of the ACM*, 45(8):345–348, 2002.

Keti Korini and Christian Bizer. Column type annotation using ChatGPT. *arXiv preprint arXiv:2306.00745*, 2023.

LO Kornum and HK Raaschou Nielsen. Surface defects in drying paint films. *Progress in Organic Coatings*, 8(3):275–324, 1980.

Georgia Koutrika. Natural language data interfaces: A data access odyssey (invited talk). In *27th International Conference on Database Theory (ICDT 2024)*, pages 1–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 11:31866–31879, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

NG Kuftinova, OI Maksimychev, AV Ostroukh, AV Volosova, and EN Matukhina. Data fabric as an effective method of data management in traffic and road systems. In *2022 Systems*

of Signals Generating and Processing in the Field of on Board Communications, pages 1–4. IEEE, 2022.

Nico Kuth. Design and implementation of a programming interface for the data lake system SEDAR. Bachelor's thesis, Hochschule Niederrhein, Krefeld, Germany, September 2023. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Wilburt Labio and Hector Garcia-Molina. Efficient snapshot differential algorithms for data warehousing. In *Proc. 22th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 63–74, Mumbai (Bombay), India, 1996. Morgan Kaufmann. URL http://www.vldb.org/conf/1996/P063.PDF.

Paul Lafargue. Karl marx. https://www.marxists.org/archive/lafargue/1890/xx/marx.htm, 1890.

Sabine Landwehr-Zloch and Josef Glaß. *Innovationsmanagement der chemischen Industrie im digitalen Zeitalter*. Springer, 2020.

Doug Laney et al. 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1, 2001.

O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3C Recommendation, February 1999. URL https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Ora Lassila, J Hendler, and T Berners-Lee. The semantic web. *Scientific American*, 284(5): 34–43, 2001.

Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. Flexible RDF generation from RDF and heterogeneous data sources with SPARQL-generate. In *Proc. EKAW Satellite Events, Revised Selected Papers*, volume 10180 of *LNCS*, pages 131–135. Springer, 2016. doi: 10.1007/978-3-319-58694-6_16.

Vladimir Ilich Lenin. The chief task of our day. In Robert C. Tucker, editor, *The Lenin Anthology*, pages 433–434. W. W. Norton & Company, New York, 1975. Written circa 1918; Part IV: The Revolutionary State and Its Policies.

Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. ACM PODS*, pages 233–246, 2002. doi: 10.1145/543613.543644.

Aristotelis Leventidis, Martin Pekár Christensen, Matteo Lissandrini, Laura Di Rocco, Katja Hose, and Renée J Miller. A large scale test corpus for semantic table search. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1142–1151, 2024.

Dawei Li, Zhen Tan, and Huan Liu. Exploring large language models for feature selection: A data-centric perspective. *ACM SIGKDD Explorations Newsletter*, 26(2):44–53, 2025.

Jiagen Li, Junzi Li, Rulin Liu, Yuxiao Tu, Yiwen Li, Jiaji Cheng, Tingchao He, and Xi Zhu. Autonomous discovery of optically active chiral inorganic perovskite nanocrystals through an intelligent cloud lab. *Nature Communications*, 11(1):2046, April 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-15728-5.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023a.

Mengke Li, Naifu Yao, Sha Liu, Shouqing Li, Yongqiang Zhao, and Seong G Kong. Multisensor image fusion for automated detection of defects in printed circuit boards. *IEEE Sensors Journal*, 21(20):23390–23399, 2021.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263*, 2023b.

XinChi Li, Mingchuan Yang, Xiaoqing Xia, Kaicheng Zhang, and Kang Liu. A distributed data fabric architecture based on metadate knowledge graph. In *2022 5th International Conference on Data Science and Information Technology (DSIT)*, pages 1–7. IEEE, 2022.

Xue Li and Till Döhmen. Towards efficient data wrangling with LLMs using code generation. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*, pages 62–66, 2024.

Ziyu Li, Wenjie Zhao, Asterios Katsifodimos, and Rihan Hai. LLM-pqa: LLM-enhanced prediction query answering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 5239–5243, 2024.

Kuo-Wei Liao and Yi-Ting Lee. Detection of rust defects on steel bridge coatings via digital image recognition. *Automation in Construction*, 71:294–306, 2016.

Hannah H. Lim, John F. Curry, and Michael T. Dugger. Improved Throughput and Analysis of Scratch Test Results via Automation and Machine Learning. Technical Report SAND2022-1836, Sandia National Lab, Albuquerque, USA, February 2022.

Jimmy Lin. The lambda and the kappa. *IEEE Internet Computing*, 21(05):60–66, 2017.

Daniel Linstedt and Michael Olschimke. *Building a scalable data warehouse with data vault 2.0*. Morgan Kaufmann, 2015.

Inc. Literal AI. Chainlit, 2024. URL https://github.com/Chainlit/chainlit. Accessed: October 15, 2024.

Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, page 100761, 2022.

Lei Liu, So Hasegawa, Shailaja Keyur Sampat, Maria Xenochristou, Wei-Peng Chen, Takashi Kato, Taisei Kakibuchi, and Tatsuya Asai. Autodw: Automatic data wrangling leveraging large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2041–2052, 2024a.

Mengyi Liu and Jianqiu Xu. Nli4db: A systematic review of natural language interfaces for databases. *arXiv preprint arXiv:2503.02435*, 2025.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024b. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Pengfei Liu, Sabine Loudcher, Jérôme Darmont, Emmanuelle Perrin, Jean-Pierre Girard, and Marie-Odile Rousset. Metadata model for an archeological data lake. In *Proc. Digital Humanities Conf.*, 2020a.

Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*, 2024c.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2901–2908, 2020b.

Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. Webvowl: Web-based visualization of ontologies. In *EKAW*, 2014.

David Loshin. *Master data management*. Morgan Kaufmann, 2010.

M. C. Lovell. Data mining. *Review of Economics and Statistics*, 65(1):1–11, 1983.

P. Lu, X. Xiao, M. Lukitsch, A. Sachdev, and Y. K. Chou. Interface characterizations of diamond-coated tools by scratch testing and simulations. *Surface and Coatings Technology*, 206(7): 1860–1866, December 2011. ISSN 0257-8972. doi: 10.1016/j.surfcoat.2011.08.022.

Elias Lumer, Vamse Kumar Subbiah, James A Burke, Pradeep Honaganahalli Basavaraju, and Austin Huber. Toolshed: Scale tool-equipped agents with advanced rag-tool fusion and tool knowledge bases. *arXiv preprint arXiv:2410.14594*, 2024.

Hanseok Ma and Sunggeun Lee. Smart system to detect painting defects in shipyards: Vision ai and a deep-learning approach. *Applied Sciences*, 12(5):2412, 2022.

Inês Araújo Machado, Carlos Costa, and Maribel Yasmina Santos. Data mesh: Concepts and principles of a paradigm shift in data architectures. In Maria Manuela Cruz-Cunha, Ricardo Martinho, Rui Rijo, Dulce Domingos, and Emanuel Peres, editors, *CENTERIS 2021 - International Conference on ENTERprise Information Systems*, volume 196 of *Procedia Computer Science*, pages 263–271. Elsevier, 2021. doi: 10.1016/J.PROCS.2021.12.013. URL https://doi.org/10.1016/j.procs.2021.12.013.

Aurora Macías, David Muñoz, Elena Navarro, and Pascual González. Data fabric and digital twins: An integrated approach for data fusion design and evaluation of pervasive systems. *Information Fusion*, 103:102–139, 2024.

Mark Madsen. *How to Build an Enterprise Data Lake: Important Considerations before Jumping*. Third Nature Inc., San Mateo, CA, USA, 2015.

Simon Malberg, Edoardo Mosca, and Georg Groh. Felix: Automatic and interpretable feature engineering using LLMs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 230–246. Springer, 2024.

Mohamed Nadjib Mami, Simon Scerri, Sören Auer, and Maria-Esther Vidal. Towards semantification of big data technology. In *Proc. DaWaK 2016*, volume 9829 of *LNCS*, pages 376–390. Springer, 2016. doi: 10.1007/978-3-319-43946-4_25.

Mohamed Nadjib Mami, Hajira Jabeen, and Sören Auer. "teach me to fish": Querying semantic data lakes. https://www.researchgate.net/publication/322526357_'Teach_me_to_fish'_Querying_Semantic_Data_Lakes, 2018. Accessed on 04.11.2024.

Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Sören Auer, and Jens Lehmann. Squerall: Virtual ontology-based access to heterogeneous and large data sources. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 229–245. Springer, 2019a.

Mohamed Nadjib Mami, Damien Graux, Harsh Thakkar, Simon Scerri, Sören Auer, and Jens Lehmann. The query translation landscape: a survey. *CoRR*, abs/1910.03118, 2019b. URL http://arxiv.org/abs/1910.03118.

F. Manola, E. Miller, and B. McBride. RDF primer. W3C Recommendation, February 2004. URL https://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

David Marco. Building and managing the meta data repository. *A full lifecycle guide*, 2000.

Alexander Martin. Design and implementation of a generic ingestion interface with versioning for data lake systems. Master's thesis, Hochschule Niederrhein, Krefeld, Germany, January 2022. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Ignacio D Martinez-Casanueva, Luis Bellido, Daniel González-Sánchez, and Diego Lopez. Candil: A federated data fabric for network analytics. *Future Generation Computer Systems*, 158:98–109, 2024.

Mary E McKnight and Jonathan W Martin. Advanced methods and models for describing coating appearance. *Progress in organic coatings*, 34(1-4):152–159, 1998.

Ben De Meester, Wouter Maroy, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. Declarative data transformations for linked data generation: The case of DBpedia. In *Proc. ESWC*, volume 10250 of *LNCS*, pages 33–48, 2017. doi: 10.1007/978-3-319-58451-5_3.

Pablo N Mendes, Max Jakob, and Christian Bizer. *DBpedia: A multilingual cross-domain knowledge base*. European Language Resources Association (ELRA), 2012.

Lars-Peter Meyer, Johannes Frey, Kurt Junghanns, Felix Brei, Kirill Bulert, Sabine Gründer-Fahrer, and Michael Martin. Developing a scalable benchmark for assessing large language models in knowledge graph engineering. *arXiv preprint arXiv:2308.16622*, 2023a.

Lars-Peter Meyer, Claus Stadler, Johannes Frey, Norman Radtke, Kurt Junghanns, Roy Meissner, Gordian Dziwis, Kirill Bulert, and Michael Martin. LLM-assisted knowledge graph engineering: Experiments with ChatGPT. *arXiv preprint arXiv:2307.06917*, 2023b.

Franck Michel, Catherine Faron-Zucker, and Johan Montagnat. Bridging the semantic web and NoSQL worlds: Generic SPARQL query translation and application to mongodb. *Trans. Large Scale Data Knowl. Centered Syst.*, 40:125–165, 2019.

William K Michener and Matthew B Jones. Ecoinformatics: supporting ecology as a data-intensive science. *Trends in ecology & evolution*, 27(2):85–93, 2012.

Libby Miller, Andy Seaborne, and Alberto Reggiori. Three implementations of squishql, a simple RDF query language. In Ian Horrocks and James A. Hendler, editors, *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*, pages 423–435. Springer, 2002. doi: 10.1007/3-540-48005-6_36. URL https://doi.org/10.1007/3-540-48005-6_36.

Renée J Miller, Mauricio A Hernández, Laura M Haas, Lingling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: managing heterogeneity. *ACM Sigmod Record*, 30 (1):78–83, 2001.

S. Mirabedini, S. Jamali, M. Haghayegh, M Sharifi, AS Mirabedini, and R Hashemi-Nasab. Application of mixture experimental design to optimize formulation and performance of thermoplastic road markings. *Progress in Organic Coatings*, 75(4):549–559, 2012.

Niels Mittelstädt. An agent based natural language interface for the semantic data lake system SEDAR. Master's thesis, RWTH Aachen University, Chair of Computer Science 5, Information Systems & Databases, Aachen, Germany, May 2025. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Advisor: Sayed Hoseini, M.Sc.

ABM Moniruzzaman and Syed Akhter Hossain. NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

Seok-Jae Moon, Seong-Beom Kang, and Byung-Joon Park. A study on a distributed data fabric-based platform in a multi-cloud environment. *International Journal of Advanced Culture Technology*, 9(3):321–326, 2021.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. MTEB: Massive text embedding benchmark, 2023. URL https://arxiv.org/abs/2210.07316.

Gerrit Muller. A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper*, page 24, 2008.

Sergi Nadal, Kashif Rabbani, Oscar Romero, and Shumet Tadesse. ODIN: A dataspace management system. In *ISWC Satellite Tracks*, volume 2456 of *CEUR WS*, pages 185–188, 2019. URL http://ceur-ws.org/Vol-2456/paper48.pdf.

Athira Nambiar and Divyansh Mundra. An overview of data warehouse and data lake in modern enterprise data management. *Big data and cognitive computing*, 6(4):132, 2022.

Aditya Nandy, Chenru Duan, Michael G. Taylor, Fang Liu, Adam H. Steeves, and Heather J. Kulik. Computational discovery of transition-metal complexes: From high-throughput screening to machine learning. *Chemical Reviews*, 121(16):9927–10000, 2021. doi: 10.1021/acs.chemrev.1c00347.

Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911*, 2022.

Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. Data lake management: challenges and opportunities. *Proc. VLDB Endow.*, 12(12):1986–1989, aug 2019. ISSN 2150-8097.

Sebastian Neumaier, Axel Polleres, Simon Steyskal, and Jürgen Umbrich. Data integration for open data on the web. *Reasoning Web. Semantic Interoperability on the Web: 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures 13*, pages 1–28, 2017.

Andrew Nguonly, Armando Magalhães, Obi-Ike Nwoke, Shervin Afshar, Sreyashi Das, Tongliang Liu, Wei Liu, and Yucheng Zeng. Data movement in netflix studio via data mesh. Netflix TechBlog, July 2021. URL https://netflixtechblog.com/data-movement-in-netflix-studio-via-data-mesh-3fddcceb1059. Accessed on 06.02.2025.

Natalya F Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W Fergerson, and Mark A Musen. Creating semantic web contents with protege-2000. *IEEE intelligent systems*, 16(2): 60–71, 2001.

Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019.

Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. Nomic embed: Training a reproducible long context text embedder. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=IPmzyQSiQE. Reproducibility Certification.

Marcelo Iury S. Oliveira and Bernadette Farias Lóscio. What is a data ecosystem? In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*, dg.o '18, New York, NY, USA, 2018. ACM. ISBN 9781450365260. doi: 10.1145/3209281.3209335. URL https://doi.org/10.1145/3209281.3209335.

OpenAI. Hello gpt-4o, 2024a. URL https://openai.com/index/hello-gpt-4o/. Accessed: October 5, 2024.

OpenAI. Introducing openai o1, 2024b. URL https://openai.com/o1/. Accessed: March 12, 2025.

Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q Pu, and Renée J Miller. Ronin: data lake exploration. *Proceedings of the VLDB Endowment*, 14(12), 2021.

M Tamer Özsu, Patrick Valduriez, et al. *Principles of distributed database systems*, volume 2. Springer, 1999.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024. doi: 10.1109/TKDE.2024.3352100.

S Peter Pappas. Weathering of coatings-formulation and evaluation. *Progress in organic coatings*, 17(2):107–114, 1989.

Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. Lotus: Enabling semantic queries with LLMs over tables of unstructured and structured data. *arXiv preprint arXiv:2407.11418*, 2024.

Pratik Patel, Gordon Wood, and Alberto Diaz. Data lake governance best practices. *Dzone Guide Big Data—Data Sci. Adv. Anal.*, 4:6–7, 2017.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Norman W Paton, Jiaoyan Chen, and Zhenyu Wu. Dataset discovery and exploration: A survey. *ACM Computing Surveys*, 56(4):1–37, 2023.

Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

Alexander Paulus. *Semi-Automated Refinement of Semantic Models*. Dissertation, Bergische Universität Wuppertal, Faculty of Electrical Engineering, Information Technology, and Media Technology, Wuppertal, Germany, 2024.

Alexander Paulus, André Pomp, Lucian Poth, Johannes Lipp, and Tobias Meisen. Gathering and combining semantic concepts from multiple knowledge bases. In *Proc. ICEIS*, pages 69–80, 2018. doi: 10.5220/0006800700690080.

Alexander Paulus, Andreas Burgdorf, André Pomp, and Tobias Meisen. Recent advances and future challenges of semantic modeling. In *Proc. 15th IEEE ICSC*, pages 70–75. IEEE, 2021a. doi: 10.1109/ICSC50631.2021.00016.

Alexander Paulus, Andreas Burgdorf, Lars Puleikis, Tristan Langer, André Pomp, and Tobias Meisen. PLASMA: platform for auxiliary semantic modeling approaches. In *Proc. ICEIS*, pages 403–412. SCITEPRESS, 2021b. doi: 10.5220/0010499604030412.

Alexander Paulus, André Pomp, and Tobias Meisen. The PLASMA framework: Laying the path to domain-specific semantics in dataspaces. In Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben, editors, *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 1474–1479. ACM, 2023. doi: 10.1145/3543873.3587662. URL https://doi.org/10.1145/3543873.3587662.

Adrian Payne and Pennie Frow. A strategic framework for customer relationship management. *Journal of marketing*, 69(4):167–176, 2005.

Ralph Peeters and Christian Bizer. Using ChatGPT for entity matching. In *European Conference on Advances in Databases and Information Systems*, pages 221–230. Springer, 2023.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*, 2024.

Minh Pham, Suresh Alse, Craig A. Knoblock, and Pedro A. Szekely. Semantic labeling: A domain-independent approach. In *Proc. ISWC*, volume 9981 of *LNCS*, pages 446–462, 2016.

Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Andriy Nikolov, Andreas Schwarte, Christian Heupel, and Tim Kraska. Incmap: A journey towards ontology-based data integration. In *Proc. BTW*, volume P-265 of *LNI*, pages 145–164. GI, 2017. URL https://dl.gi.de/20.500.12116/625.

Guil Pires, Mark Cho, Mingliang Liu, and Sujay Jain. Streaming SQL in data mesh. Netflix TechBlog, November 2023. URL https://netflixtechblog.com/streaming-sql-in-data-mesh-0d83f5a00d08. Accessed on 06.02.2025.

Plattform Industrie 4.0. Project gaia-x: A federated data infrastructure as the cradle of a vibrant european ecosystem. Technical report, Federal Ministry for Economic Affairs and Energy (BMWi), D-11019 Berlin, Germany, Oct. 2019.

Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semant.*, 10:133–173, 2008.

Dominik Polke, Florian Diepers, Elmar Ahle, and Dirk Söffker. Development of a framework for data-driven modeling with cloud services in the process industry. In *2022 International Conference on Automation, Robotics and Computer Engineering (ICARCE)*, pages 109–115, 2022. doi: 10.1109/ICARCE55724.2022.10046584.

Dominik Polke, Alvin Surjana, Florian Diepers, Elmar Ahle, and Dirk Söffker. Development of a modular automation framework for data-driven modeling and optimization of coating formulations. In *28th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2023, Sinaia, Romania, September 12-15, 2023*, pages 1–8. IEEE, 2023. doi: 10.1109/ETFA54631.2023.10275680.

Dominik Polke, Alvin Surjana, Gaoyuan Zhang, Sayed Hoseini, Lasse Wagner, Carmen Stoffelen-Janßen, Christian Schmitz, Jost Göttert, Christoph Quix, and Elmar Ahle. Entwicklung eines automatisierungsframeworks zur mehrgrößenoptimierung in chemischen hochdurchsatz-prozessen. *VDI AUTOMATION 2024*, 2024.

Dominik Polke, Alvin Immanuel Surjana, Sayed Hoseini, Lasse Wagner, Jost Gottert, Christoph Quix, and Elmar Ahle. Development of an automation framework for data-driven modeling and adaptive learning in industrial processes. In *Proceedings of the 9th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2026.

Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1723–1726. ACM, 2017. doi: 10.1145/3035918.3054782. URL https://doi.org/10.1145/3035918.3054782.

André Pomp, Alexander Paulus, Sabina Jeschke, and Tobias Meisen. ESKAPE: information platform for enabling semantic data processing. In *Proc. ICEIS*, pages 644–655. SciTePress, 2017. doi: 10.5220/0006324906440655.

André Pomp, Alexander Paulus, Andreas Kirmse, Vadim Kraus, and Tobias Meisen. Applying semantics to reduce the time to analytics within complex heterogeneous infrastructures. *Technologies*, 6(3):86, 2018.

André Pomp, Johannes Lipp, and Tobias Meisen. You are missing a concept! enhancing ontology-based data access with evolving ontologies. In *Proc. ICSC*, pages 98–105. IEEE, 2019a. doi: 10.1109/ICOSC.2019.8665620.

André Pomp, Johannes Lipp, and Tobias Meisen. Enabling the continuous evolution of ontologies for ontology-based data management. *Intl. Journal of Robotic Computing*, 2019b.

André Pomp, Alexander Paulus, Andreas Burgdorf, and Tobias Meisen. A semantic data marketplace for easy data sharing within a smart city. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, page 4774–4778, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3481995. URL https://doi.org/10.1145/3459637.3481995.

Aleš Popovič, Ray Hackney, Rana Tassabehji, and Mauro Castelli. The impact of big data analytics on firms' high value business performance. *Information Systems Frontiers*, 20: 209–222, 2018.

Gil Press. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says, 2016. URL https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says. [Accessed: 07-Oct-2024].

Torsten Priebe, Sebastian Neumaier, and Stefan Markus. Finding your way through the jungle of big data architectures. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5994–5996, 2021. doi: 10.1109/BigData52589.2021.9671862.

Edward O. Pyzer-Knapp, Jed W. Pitera, Peter W. J. Staar, Seiji Takeda, Teodoro Laino, Daniel P. Sanders, James Sexton, John R. Smith, and Alessandro Curioni. Accelerating materials discovery using artificial intelligence, high performance computing and robotics. *npj Computational Materials*, 8(1):1–9, April 2022. ISSN 2057-3960. doi: 10.1038/s41524-022-00765-z. Number: 1 Publisher: Nature Publishing Group.

Pascal Quindeau, Sayed Hoseini, Jonas Kaufmann, and Christoph Quix. Einsatz einer webbasierten softwarelösung zur echtzeitkollaboration in technisch-orientierten studienfächern. 2022.

Christoph Quix and Rihan Hai. Data lake. In *Encyclopedia of Big Data Technologies*. Springer, 2019. doi: 10.1007/978-3-319-63962-8_7-1.

Christoph Quix, Rihan Hai, and Ivan Vatov. Metadata extraction and management in data lakes with GEMMS. *Complex Syst. Informatics Model. Q.*, 2016.

Christoph Quix, Elmar Ahle, Jost Göttert, and Christian Schmitz. Proposal for a single project under the directive Research at Universities of Applied Sciences in Cooperation with Industry (FH-Kooperativ): Intelligent infrastructure for data management in automated chemical high-throughput processes (i2dach). Project proposal, Hochschule Niederrhein, 2023.

Christoph Josef Quix. *Metadatenverwaltung zur qualitätsorientierten Informationslogistik in Data-Warehouse-Systemen*. PhD thesis, Bibliothek der RWTH Aachen, 2003.

Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10:334–350, 2001.

Raghu Ramakrishnan and Johannes Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003. ISBN 978-0-07-115110-8.

Raghu Ramakrishnan, Baskar Sridharan, John R Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, et al. Azure data lake store: a hyperscale distributed file service for big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 51–63, 2017.

S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro A. Szekely. Assigning semantic labels to data sources. In *Proc. ESWC*, volume 9088 of *LNCS*, pages 403–417. Springer, 2015. doi: 10.1007/978-3-319-18818-8_25.

Nicholas X. Randall. The current state-of-the-art in scratch testing of coated systems. *Surface and Coatings Technology*, 380, December 2019. ISSN 0257-8972. doi: 10.1016/j.surfcoat.2019.125092.

Tye Rattenbury, Joseph M Hellerstein, Jeffrey Heer, Sean Kandel, and Connor Carreras. *Principles of data wrangling: Practical techniques for data preparation*. " O'Reilly Media, Inc.", 2017.

Kenneth G. Rau. Effective governance of it: Design objectives, roles, and relationships. *Inf. Syst. Manag.*, 21(4):35–42, 2004. doi: 10.1201/1078/44705.21.4.20040901/84185.4. URL https://doi.org/10.1201/1078/44705.21.4.20040901/84185.4.

Franck Ravat and Yan Zhao. Data lakes: Trends and perspectives. In *Database and Expert Systems Applications: 30th International Conference, DEXA 2019, Linz, Austria, August 26–29, 2019, Proceedings, Part I 30*, pages 304–313. Springer, 2019.

Vamsee Krishna Ravi and Aravind Ayyagari. Data lake implementation in enterprise environments. *International Journal of Progressive Research in Engineering Management and Science (IJPREMS) Vol*, 3, 2023.

RDF Schema 1.1. RDF Schema 1.1. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/, February 2014.

Thomas Rebele, Fabian Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*, pages 177–185. Springer, 2016.

Thomas C Redman. *Data quality: the field guide*. Digital press, 2001.

Nils Reimers and Iryna Gurevych. Sentence transformers: Embeddings, retrieval, and reranking, 2025. URL https://github.com/UKPLab/sentence-transformers. Accessed: March 28, 2025.

Navid Rekabsaz, Mihai Lupu, and Allan Hanbury. Exploration of a threshold for similarity based on uncertainty in word embedding. In Joemon M Jose, Claudia Hauff, Ismail Sengor Altıngovde, Dawei Song, Dyaa Albakour, Stuart Watt, and John Tait, editors, *Advances in Information Retrieval*, pages 396–409, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56608-5.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis, 2020. URL https://arxiv.org/abs/2009.10297.

Sakib Anwar Rieyan, Md Raisul Kabir News, ABM Muntasir Rahman, Sadia Afrin Khan, Sultan Tasneem Jawad Zaarif, Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Michele Ianni, and Giancarlo Fortino. An advanced data fabric architecture leveraging homomorphic encryption and federated learning. *Information Fusion*, 102:102004, 2024.

RML Community. RML: The RDF Mapping Language. Specification / Documentation. URL https://rml.io/specs/rml/.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. URL https://doi.org/10.1007/978-3-319-24574-4_28.

Jennifer Rowley. The wisdom hierarchy: representations of the dikw hierarchy. *Journal of information science*, 33(2):163–180, 2007.

Thomas A Runkler. *Data analytics*. Springer, 2020.

Go Muan Sang, Lai Xu, and Paul de Vrieze. A reference architecture for big data systems. In *2016 10th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 370–375, 2016. doi: 10.1109/SKIMA.2016.7916249.

Go Muan Sang, Lai Xu, and Paul De Vrieze. Simplifying big data analytics systems with a reference architecture. In *Collaboration in a Data-Rich World: 18th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2017, Vicenza, Italy, September 18-20, 2017, Proceedings 18*, pages 242–249. Springer, 2017.

Aécio Santos, Eduardo HM Pena, Roque Lopez, and Juliana Freire. Interactive data harmonization with LLM agents. *arXiv preprint arXiv:2502.07132*, 2025.

Pegdwendé Sawadogo and Jérôme Darmont. On data lake architectures and metadata management. *JJIS*, 2021.

Pegdwendé N Sawadogo, Etienne Scholly, Cécile Favre, Eric Ferey, Sabine Loudcher, and Jérôme Darmont. Metadata systems for data lakes: models and features. In *European conference on advances in databases and information systems*, pages 440–451. Springer, 2019.

Sebastian Schelter, Felix Biessmann, Dustin Lange, Tammo Rukat, Phillipp Schmidt, Stephan Seufert, Pierre Brunelle, and Andrey Taptunov. Unit testing data with deequ. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1993–1996, 2019.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36: 68539–68551, 2023.

Marius Schlegel and Kai-Uwe Sattler. Extracting provenance of machine learning experiment pipeline artifacts. In *27th ADBIS Conference, Barcelona, Spain*, volume 13985 of *LNCS*, pages 238–251. Springer, 2023.

Christian Schmitz, Kevin Cremanns, and Golnaz Bissadi. Application of machine learning algorithms for use in material chemistry. In *Computational and Data-Driven Chemistry Using Artificial Intelligence*, pages 161–192. Elsevier, 2022.

Edward W Schneider. Course modularization applied: The interface system and its implications for sequence control and data analysis. 1973.

Jan Schneider, Christoph Gröger, Arnold Lutsch, Holger Schwarz, and Bernhard Mitschang. Assessing the lakehouse: Analysis, requirements and definition. In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023)*. Scitepress 2023, 2023.

Jan Schneider, Christoph Gröger, Arnold Lutsch, Holger Schwarz, and Bernhard Mitschang. The lakehouse: State of the art on concepts and technologies. *SN Computer Science*, 5(5): 1–39, 2024.

O Schneider, GO Ilevbare, JR Scully, and RG Kelly. Confocal laser scanning microscopy as a tool for in situ monitoring of corrosion underneath organic coatings. *Electrochemical and Solid-State Letters*, 4(12):B35, 2001.

Clifford K Schoff and Schoff Associates. Optical microscopy. *JCT coatingstech*, 3(2):36–43, 2006.

Etienne Scholly, Pegdwendé Sawadogo, Pengfei Liu, Javier Alfonso Espinosa-Oviedo, Cécile Favre, Sabine Loudcher, Jérôme Darmont, and Camille Noûs. Coining goldmedal: a new contribution to data lake generic metadata modeling. *arXiv preprint arXiv:2103.13155*, 2021.

Elske M. Schönhals and Sayed Hoseini. Is the data awareness rally really fun? exploring assumptions in course design by analysing student feedback. In *Proceedings of the PEDaL HN Project, Niederrhein University*, 2025. doi: https://doi.org/10.82523/HSNR-516.

G. Schreiber and Y. Raimond. RDF 1.1 primer. W3C Working Group Note, June 2014. URL https://www.w3.org/TR/rdf11-primer/.

Yannik Schuchmann, Christian Schmitz, Lasse Wagner, Carmen Stoffelen-Janßen, and Jost Göttert. Automated modelling of material chemistry applying high-throughput formulation and machine learning. *Modern Concepts in Material Science*, 3(4), 2021. doi: 10.33552/MCMS.2021.03.000570. URL https://doi.org/10.33552/MCMS.2021.03.000570.

Max Schultze and Arif Wider. Data mesh in practice: How europe's leading online platform for fashion goes beyond the data lake. Online video, 2020. Available at: https://www.youtube.com/watch?v=eiUhV56uVUc [Accessed 06.02.2025].

Michael Schulz, Uwe Neuhaus, Jens Kaufmann, Stephan Kühnel, Emal M Alekozai, Heiko Rohde, Sayed Hoseini, René Theuerkauf, Daniel Badura, Ulrich Kerzel, et al. Dasc-pm v1. 1 a process model for data science projects. 2022.

Michael Schulz, Uwe Neuhaus, Stephan Kühnel, Heiko Rohde, Sayed Hoseini, and René Theuerkauf. Dasc-pm v1. 1 case studies, 2023.

Arun Sen. Metadata management: past, present and future. *Decis. Support Syst.*, 37(1): 151–173, 2004. doi: 10.1016/S0167-9236(02)00208-7. URL https://doi.org/10.1016/S0167-9236(02)00208-7.

James Serra. *Deciphering Data Architectures*. O'Reilly Media, Inc., 2024.

Haron Shaker. Generation of semantic models in a semantic data lake using gnns. Master's thesis, RWTH Aachen University, Aachen, Germany, October 2023. Supervised by Prof. Dr. Stefan Decker, second supervisor: Prof. Dr. Christoph Quix, advisor: Sayed Hoseini, M.Sc.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Anmol Sharma. LLM-based tool for FAIR data assessment. Master's thesis, RWTH Aachen University, Chair of Computer Science 5, Information Systems & Databases, Aachen, Germany, May 2025. Supervisors: Prof. Dr. Stefan Decker, Prof. Dr. Christoph Quix. Co-Advisor: Sayed Hoseini, M.Sc.

Anmol Sharma, Sulayman K. Sowe, Soo-Yon Kim, Sayed Hoseini, Fidan Limani, Zeyd Boukhers, Christoph Lange, and Stefan Decker. FAIR data assessment using LLMs: The FAIR-way. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. ACM, 2025.

Ben Sharma. *Architecting Data Lakes—Data Management Architectures for Advanced Business Use Cases*. O'Reilly Media, Sebastopol, CA, USA, 2nd edition, 2018.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with ChatGPT and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.

Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, September 1990. ISSN 0360-0300. doi: 10.1145/96602.96604. URL https://doi.org/10.1145/96602.96604.

Álvaro Sicilia, German Nemirovski, and Andreas Nolle. Map-on: A web-based editor for visual ontology mapping. *Semantic Web*, 8(6):969–980, 2017. doi: 10.3233/SW-160246.

Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. *Database system concepts*, volume 5. McGraw-Hill New York, 2002.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Umutcan Simsek, Elias Kärle, and Dieter Fensel. Rocketrml - A nodejs implementation of a use case specific RML mapper. In *Proc. ESWC Workshops*, volume 2489 of *CEUR WS*, pages 46–53, 2019. URL http://ceur-ws.org/Vol-2489/paper5.pdf.

Maulshree Singh, Evert Fuenmayor, Eoin P Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future. *Applied System Innovation*, 4(2):36, 2021.

CData Software. Data fabric vs. data virtualization, 2024. URL https://www.cdata.com/blog/data-fabric-vs-data-virtualization. Accessed: 2024-11-22.

Gürkan Solmaz, Flavio Cirillo, Jonathan Fürst, Tobias Jacobs, Martin Bauer, Ernö Kovacs, Juan Ramón Santana, and Luis Sánchez. Enabling data spaces: existing developments and challenges. In *Proceedings of the 1st International Workshop on Data Economy*, DE '22, page 42–48, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450399234. doi: 10.1145/3565011.3569058.

Nicolae Solomon and Iulia Solomon. Paint blistering affects quality of finished products. *Engineering Failure Analysis*, 92:44–49, 2018.

Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In Andreas Thalhammer, Gong Cheng, and Kalpa Gunaratna, editors, *Proceedings of the 2nd International Workshop on Summarizing and Presenting Entities and Ontologies (SumPre 2016) co-located with the 13th Extended Semantic Web Conference (ESWC 2016), Anissaras, Greece, May 30, 2016*, volume 1605 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL https://ceur-ws.org/Vol-1605/paper3.pdf.

Robyn Speer, Catherine Havasi, et al. Representing general relational knowledge in conceptnet 5. In *LREC*, volume 2012, pages 3679–86, 2012.

Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, and Niklas Lindström. *JSON-LD 1.1–a JSON-based serialization for Linked Data*. PhD thesis, W3C, 2019.

Marika Spring, Camilla Ricci, David A Peggie, and Sergei G Kazarian. Atr-ftir imaging for the analysis of organic materials in paint cross sections: case studies on paint samples from the national gallery, london. *Analytical and bioanalytical chemistry*, 392(1):37–45, 2008.

Ulf Stalmach. Wie funktioniert eine smart paint factory? *JOT Journal für Oberflächentechnik*, 59(9):68–73, 2019.

P. A. Steinmann, Y. Tardy, and H. E. Hintermann. Adhesion testing by the scratch test method: The influence of intrinsic and extrinsic parameters on the critical load. *Thin Solid Films*, 154 (1):333–349, November 1987. ISSN 0040-6090. doi: 10.1016/0040-6090(87)90377-4.

Bernd Strehmel, Christian Schmitz, Kevin Cremanns, and Jost Göttert. Photochemistry with cyanines in the near infrared: A step to Chemistry 4.0 technologies. *Chemistry–A European Journal*, 25(56):12855–12864, 2019.

Hans-Joachim Streitberger and Artur Goldschmidt. *BASF Handbook Basics of Coating Technology*. European Coatings, 2018.

Piethein Strengholt. *Data Management at Scale*. " O'Reilly Media, Inc.", 2023.

Rudi Studer, V Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

Pranav Subramaniam, Udayan Khurana, Kavitha Srinivas, and Horst Samulowitz. Related table search for numeric data using large language models and enterprise knowledge graphs. In *EKG-LLM@ CIKM*, 2023.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1493–1503, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517906.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. A survey on large language model-based agents for statistics and data science. *The American Statistician*, pages 1–14, 2025.

Ilya Sutskever. Sequence to sequence learning with neural networks: What a decade. https://www.youtube.com/watch?v=WQQdd6qGxNs, access on 19.02.2025, 2024. NeurIPS 2024.

Pedro A. Szekely, Craig A. Knoblock, Jason Slepicka, Andrew Philpot, Amandeep Singh, Chengye Yin, Dipsy Kapoor, Prem Natarajan, Daniel Marcu, Kevin Knight, David Stallard, Subessware S. Karunamoorthy, Rajagopal Bojanapalli, Steven Minton, Brian Amanatullah, Todd Hughes, Mike Tamayo, David Flynt, Rachel Artiss, Shih-Fu Chang, Tao Chen, Gerald Hiebel, and Lidia Silva Ferreira. Building and using a knowledge graph to combat human trafficking. In *Proc. ISWC*, volume 9367 of *LNCS*, pages 205–221. Springer, 2015. doi: 10.1007/978-3-319-25010-6_12.

Mohsen Taheriyan, Craig A. Knoblock, Pedro Szekely, Jose Luis Ambite, and Yinyi Chen. Leveraging Linked Data to Infer Semantic Relations within Structured Sources. In Olaf Hartic, Juan Sequeda, and Aidan Hogan, editors, *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015)*. 2015.

Yuxing Tan, Liang He, Xiaozhe Qi, Zhida Song, and Tengfei Feng. Tabmeta: A novel framework for chart generation using metadata and enhanced column attention. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2025.

Xiu Tang, Wenhao Liu, Sai Wu, Chang Yao, Gongsheng Yuan, Shanshan Ying, and Gang Chen. Queryartisan: Generating data manipulation codes for ad-hoc analysis in data lakes. *Proc. VLDB Endow.*, 18(2):108–116, October 2024. ISSN 2150-8097. doi: 10.14778/3705829.3705832. URL https://doi.org/10.14778/3705829.3705832.

Stackable Data Platform Team. Interview on the stackable data platform. Interview with the Stackable team, conducted by the author, May 2025. Personal communication via phone interview.

Carol Tenopir, Natalie M Rice, Suzie Allard, Lynn Baird, Josh Borycz, Lisa Christian, Bruce Grant, Robert Olendorf, and Robert J Sandusky. Data sharing, management, use, and reuse: Practices and perceptions of scientists worldwide. *PloS one*, 15(3):e0229003, 2020.

Robert Thanaraj, Mark Beyer, and Ehtisham Zaidi. What is data fabric design? https://info.cambridgesemantics.com/hubfs/What_Is_Data_Fabric_Gartner.pdf, 2021. Accessed on 22.11.2024.

The Data Governance Institute. Definitions of Data Governance, 2024a. URL https://datagovernance.com/the-data-governance-basics/definitions-of-data-governance/. [Accessed: 07. October 2025].

The Data Governance Institute. Definitions of Data Governance, 2024b. URL https://datagovernance.com/the-data-governance-basics/. [Accessed: 07. October 2025].

Johannes Theissen-Lipp. *Semantic Foundations of Dataspaces*. Dissertation, RWTH Aachen University, Faculty of Mathematics, Computer Science and Natural Sciences, 2024.

René Theuerkauf, Stephan Daurer, Sayed Hoseini, Jens Kaufmann, Stephan Kühnel, Florian Schwade, Emal M Alekozai, Uwe Neuhaus, Heiko Rohde, and Michael Schulz. Vorschlag

eines morphologischen kastens zur charakterisierung von data-science-projekten. *Informatik Spektrum*, 45(6):395–401, 2022.

Marcel Thiel. Development of a semantic metadata management system and its integration into a data lake prototype system. Master's thesis, Hochschule Niederrhein, Krefeld, Germany, May 2022. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

M. Thomson and C. Benfield. Hypertext transfer protocol (http/2). RFC 9113, Standards Track, IETF, June 2022. URL https://www.rfc-editor.org/info/rfc9113.

Carlo Tomasi and Takeo Kanade. Detection and tracking of point. *Int J Comput Vis*, 9:137–154, 1991.

Alexander Tornede et al. AutoML in the age of large language models: Current challenges, future opportunities and risks. *arXiv preprint arXiv:2306.08107*, 2023.

Milena Trajanoska, Riste Stojanov, and Dimitar Trajanov. Enhancing knowledge graph construction using large language models. *arXiv preprint arXiv:2305.04676*, 2023.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. AutoML-agent: A multi-agent LLM framework for full-pipeline AutoML. *arXiv preprint arXiv:2410.02958*, 2024.

Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950. doi: 10.1093/mind/LIX.236.433.

S. Tyagi. Using data analytics for greater profits. *Journal of Business Strategy*, 24(3):12–14, 2003.

Thomas H-J Uhlemann, Christian Lehmann, and Rolf Steinhilper. The digital twin: Realizing the cyber-physical production system for industry 4.0. *Procedia Cirp*, 61:335–340, 2017.

University of Wuppertal. Plasma physics group, university of wuppertal, 2025.

Atiya Usmani, M. Jaleed Khan, John G. Breslin, and Edward Curry. Towards multimodal knowledge graphs for data spaces. In *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, page 1494–1499, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394192. doi: 10.1145/3543873.3587665.

Alp Ustundag and Emre Cevikcan. *Industry 4.0: Managing the Digital Transformation*. Springer, 2018.

John Van den Hoven. Data architecture: Blueprints for data. *Information systems management*, 20(1), 2003.

Wil MP van der Aalst, Matthias Jarke, István Koren, and Christoph Quix. Digital shadows: Infrastructuring the internet of production. In *Internet of Production: Fundamentals, Applications and Proceedings*, pages 1–17. Springer, 2023.

Daniel van der Werf, João Moreira, and Jean Paul Sebastian Piest. Towards a data mesh reference architecture. In *International Conference on Enterprise Design, Operations, and Computing*, pages 339–353. Springer, 2024.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

VDI/VDE/NAMUR 2658 Part 1:2019-10. *Automation engineering of modular systems in the process industry – General concept and interfaces*. Beuth Verlag, Berlin, 2022.

Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, 38(1):3–es, 2006.

Binh Vu and Craig A. Knoblock. SAND : A Tool for Creating Semantic Descriptions of Tabular Sources. In Paul Groth, editor, *The semantic web*, volume 13384 of *Lecture Notes in Computer Science*, pages 63–67. Springer, Cham, 2022. ISBN 978-3-031-11608-7.

Binh Vu, Craig Knoblock, and Jay Pujara. Learning Semantic Models of Data Sources Using Probabilistic Graphical Models. In *The World Wide Web Conference*, WWW '19, pages 1944–1953, New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313711.

Binh Vu, Jay Pujara, and Craig A. Knoblock. D-REPR: A language for describing and mapping diversely-structured data sources to RDF. In Mayank Kejriwal, Pedro A. Szekely, and Raphaël Troncy, editors, *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019*, pages 189–196. ACM, 2019b. doi: 10.1145/3360901.3364449. URL https://doi.org/10.1145/3360901.3364449.

Jin Wang, Yaqiong Yang, Tian Wang, R Simon Sherratt, and Jingyu Zhang. Big data service architecture: a survey. *Journal of Internet Technology*, 21(2):393–405, 2020.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.

Kristin Weber, Boris Otto, and Hubert Österle. One size does not fit all - A contingency approach to data governance. *ACM J. Data Inf. Qual.*, 1(1):4:1–4:27, 2009. doi: 10.1145/1515693.1515696. URL https://doi.org/10.1145/1515693.1515696.

Jan-Philipp Wegmeyer. Development of a concept for data science projects in smart cities. Bachelor's thesis, Hochschule Niederrhein, Krefeld, Germany, September 2022. Supervised by Prof. Dr. Christoph Quix, second examiner: Sayed Hoseini, M.Sc.

Patric Wicht, Alexandra Hausmann, Sayed Hoseini, and Jens Kaufmann. Aufbau eines data-science-teams–„lessons learned “. *Wirtschaftsinformatik & Management*, 13(4):266–275, 2021.

Arif Wider, Sumedha Verma, and Atif Akhtar. Decentralized data governance as part of a data mesh platform: Concepts and approaches. In *2023 IEEE International Conference on Web Services (ICWS)*, pages 746–754, 2023. doi: 10.1109/ICWS60048.2023.00101.

Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Zhang Xiang and Chen Sheng-yong. A 3d image reconstruction model from multiple images. *Procedia Engineering*, 23:678–683, 2011.

Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, pages 5511–5519. ijcai.org, 2018.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. Give us the facts: Enhancing large language models with knowledge graphs for fact-aware language modeling. *IEEE Transactions on Knowledge and Data Engineering*, 2024a.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36:71995–72007, 2023.

Rui Yang, Haoran Liu, Qingcheng Zeng, Yu He Ke, Wanxin Li, Lechao Cheng, Qingyu Chen, James Caverlee, Yutaka Matsuo, and Irene Li. Kg-rank: Enhancing large language models for medical qa with knowledge graphs and ranking techniques. *arXiv preprint arXiv:2403.05881*, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Xiaojun Yu and Qiaoyan Wen. A view about cloud data security from data life cycle. In *2010 International Conference on Computational Intelligence and Software Engineering*, pages 1–4, 2010. doi: 10.1109/CISE.2010.5676895.

Yuhui Yun, Huilong Ye, Xinru Li, Ruojia Li, Jingfeng Deng, Li Li, and Haoyi Xiong. Eicopilot: Search and explore enterprise information over large-scale knowledge graphs with LLM-driven agents. *arXiv preprint arXiv:2501.13746*, 2025.

Alejandro Gabriel Villanueva Zacarias, Peter Reimann, Christian Weber, and Bernhard Mitschang. Assistml: an approach to manage, recommend and reuse ML solutions. *Int. J. Data Sci. Anal.*, 16(4):455–479, 2023. doi: 10.1007/S41060-023-00417-5. URL https://doi.org/10.1007/s41060-023-00417-5.

Matei Zaharia et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

Ehitsham Zaidi, E Thoo, G De Simoni, and M Beyer. Data fabrics add augmented intelligence to modernize your data integration. *With Eric Thoo. Gartner Grou*, 17, 2019.

Forough Zareanshahraki and Vijay Mannari. Formulation and optimization of radiation-curable nonisocyanate polyurethane wood coatings by mixture experimental design. *Journal of Coatings Technology and Research*, 18:695–715, 2021.

Abe Zeid, Sarvesh Sundaram, Mohsen Moghaddam, Sagar Kamarthi, and Tucker Marion. Interoperability in smart manufacturing: Research challenges. *Machines*, 7(2), 2019. doi: 10.3390/machines7020021.

Bo Zhang, Ansila Mathoor, and Tanja Junkers. High Throughput Multidimensional Kinetic Screening in Continuous Flow Reactors. *Angewandte Chemie International Edition*, 62(38): e202308838, 2023a. ISSN 1521-3773. doi: 10.1002/anie.202308838.

Gaoyuan Zhang, Christian Schmitz, Matthias Fimmers, Christoph Quix, and Sayed Hoseini. Deep learning-based automated characterization of crosscut tests for coatings via image segmentation. *Journal of Coatings Technology and Research*, pages 1–13, 2022.

Gaoyuan Zhang, Tobias Borgert, Carmen Stoffelen, and Christian Schmitz. High-throughput and explainable machine learning for lacquer formulations: Enhancing coating development by interpretable models. *Available at SSRN 5077294*, 2024a. doi: https://dx.doi.org/10.2139/ssrn.5077294.

Gaoyuan Zhang, Christoph Quix, Sayed Hoseini, and Christian Schmitz. Natural-language interfaces to a process-aware, graph-based ELN: An UV coatings case study. *Digital Chemical Engineering*, page 100310, 2026. ISSN 2772-5081. doi: https://doi.org/10.1016/j.dche.2026.100310.

Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979*, 2023b.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. AutoML-gpt: Automatic machine learning with gpt. *arXiv preprint arXiv:2305.02499*, 2023c.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206*, 2023d.

Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. Directions towards efficient and automated data wrangling with large language models. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*, pages 301–304. IEEE, 2024b.

Yan Zhao, Imen Megdiche, and Franck Ravat. Data lake ingestion management. *arXiv preprint arXiv:2107.02885*, 2021a.

Yan Zhao et al. Analysis-oriented metadata for data lakes. In *Proceedings of the 25th IDEAS*, page 194–203. ACM, 2021b.

Yilun Zhao, Yitao Long, Hongjun Liu, Linyong Nan, Lyuhao Chen, Ryo Kamoi, Yixin Liu, Xiangru Tang, Rui Zhang, and Arman Cohan. Docmath-eval: Evaluating numerical reasoning capabilities of LLMs in understanding long documents with tabular data. *arXiv preprint arXiv:2311.09805*, 2023.

Yongchao Zhao, Jiangang Wang, Xuejun Cui, and Hongyan Wang. The use of photoshop software to estimate the adhesion and rust-resistant properties of coating film. *Surface and interface analysis*, 43(5):913–917, 2011.

Baifan Zhou, Yulia Svetashova, Andre Gusmao, Ahmet Soylu, Gong Cheng, Ralf Mikut, Arild Waaler, and Evgeny Kharlamov. Semml: Facilitating development of ML models for condition monitoring with semantics. *J. Web Semant.*, 71:100664, 2021. doi: 10.1016/j.websem.2021.100664.

Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to ChatGPT. *International Journal of Machine Learning and Cybernetics*, pages 1–65, 2024.

Qinbang Zhou, Renwen Chen, Bin Huang, Chuan Liu, Jie Yu, and Xiaoqing Yu. An automatic surface defect inspection system for automobiles using machine vision methods. *Sensors*, 19(3):644, 2019.

Qing Zhu, Fei Zhang, Yan Huang, Hengyu Xiao, LuYuan Zhao, XuChun Zhang, Tao Song, XinSheng Tang, Xiang Li, Guo He, BaoChen Chong, JunYi Zhou, YiHan Zhang, Baicheng Zhang, JiaQi Cao, Man Luo, Song Wang, GuiLin Ye, WanJun Zhang, Xin Chen, Shuang Cong, Donglai Zhou, Huirong Li, Jialei Li, Gang Zou, WeiWei Shang, Jun Jiang, and Yi Luo. An all-round AI-Chemist with a scientific mind. *National Science Review*, 9(10):nwac190, October 2022. ISSN 2095-5138. doi: 10.1093/nsr/nwac190.

Paul Zikopoulos, Dirk DeRoos, Chris Bienko, Robert Buglio, and Mike Andrews. *Big Data Beyond the Hype*. McGraw-Hill Education, New York, NY, USA, 1st edition, 2015.

Appendix

A Scientific Equipment at the HIT Institute

A.1 HTF base system



Figure A.1: Images of the HTF equipment

A.2 GSI-Module

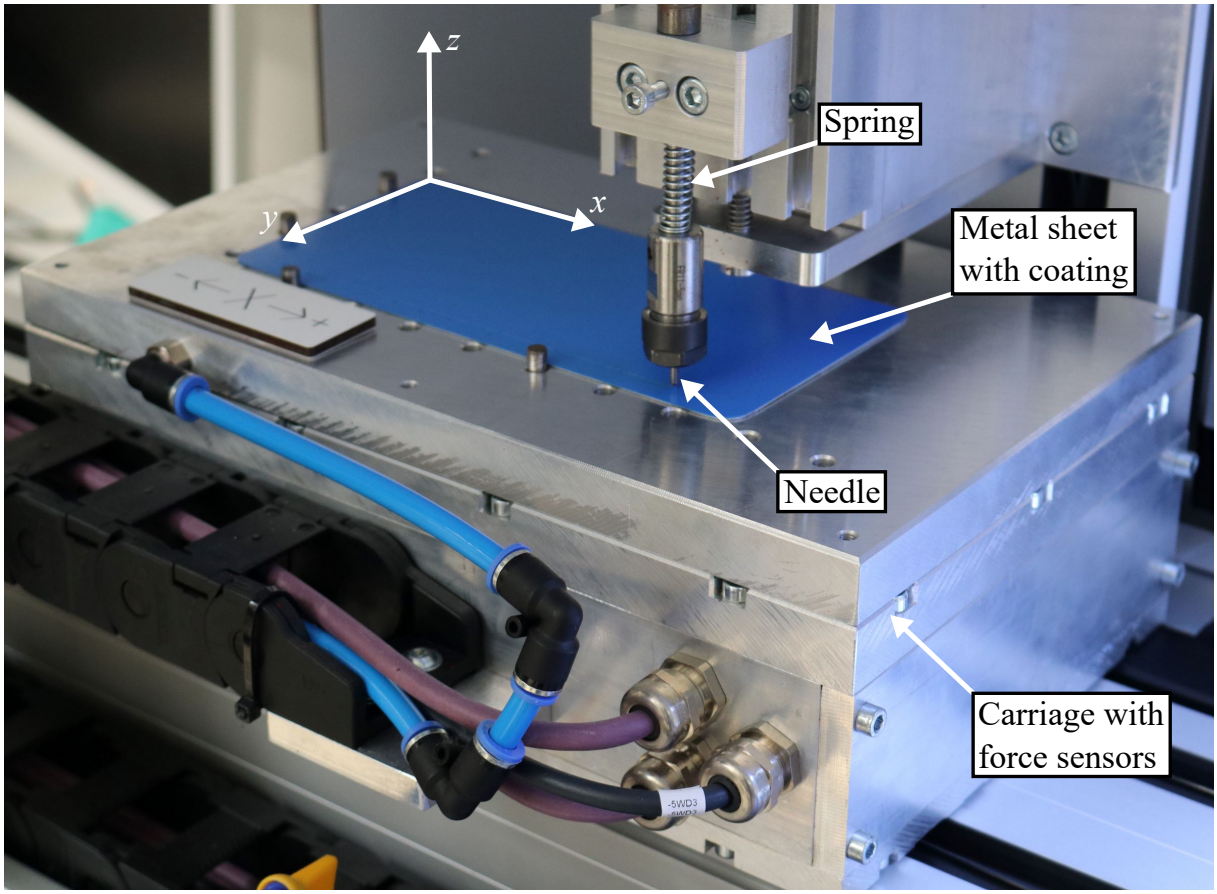


Figure A.2: Automated scratch test equipment

B SEDAR

B.1 Ingestion Workflow

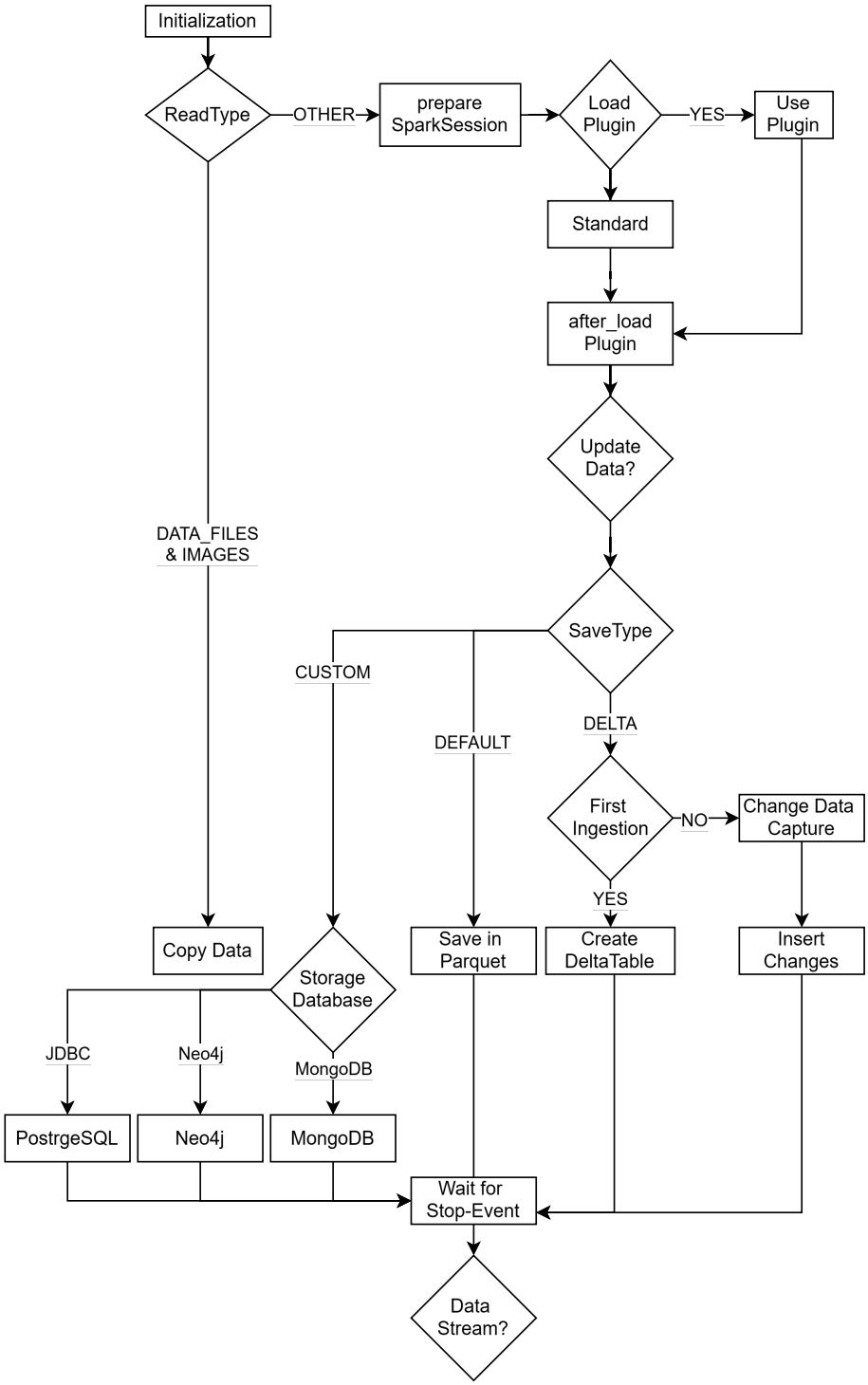


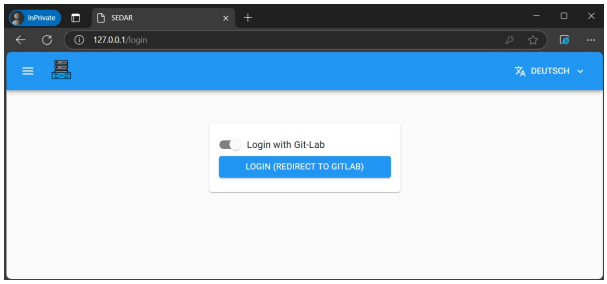
Figure B.1: Ingestion workflow as depicted by Martin [2022]

B.2 Delta calculation

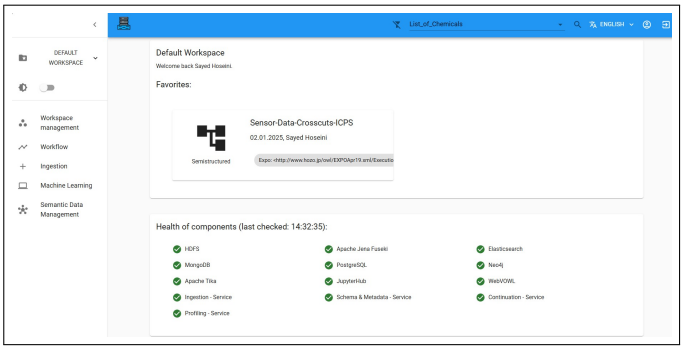
Algorithm 3: Delta calculation

```
Input: left: DataFrame, right: DataFrame, id_column: String
Output: change_data: DataFrame
foreach df in {left, right} do
    df.add_column(name: "hash", value: df.hash_over_all_rows());
    df.add_column(name: " id", value: df.get_value(id_column));
change_data ← left.join(right, type: full-outer, column: " id");
change_data.remove_all_rows(condition: left:hash equals right:hash);
change_data.add_column(name: "cd_deleted", value: false);
foreach row in change_data:rows do
    if row:right:hash is Null then
        row:cd_deleted ← true;
foreach column in left:columns do
    if column is id_column then
        foreach row in change_data:rows do
            if row:right:hash is Null then
                row:id_column ← row:left:id_column;
            else
                row:id_column ← row:right:id_column;
    else
        foreach row in change_data:rows do
            row:column ← row:right:column;
    change_data.remove_column(right:column);
    change_data.remove_column(left:column);
change_data.remove_column(" id");
change_data.remove_column("hash");
```

B.3 User Interface Screenshots

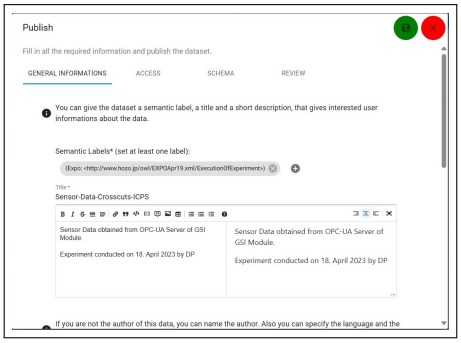


(a) SEDAR's entry page is a login form forwarding the user to login via GitLab.



(b) SEDAR's dashboard, here shown for the *Default Workspace*.

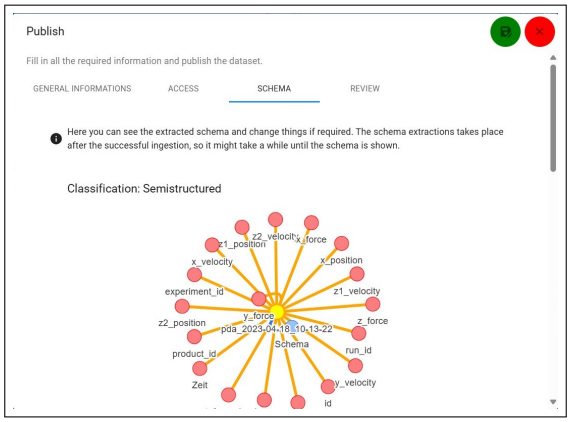
Figure B.2: Entry pages of SEDAR's UI.



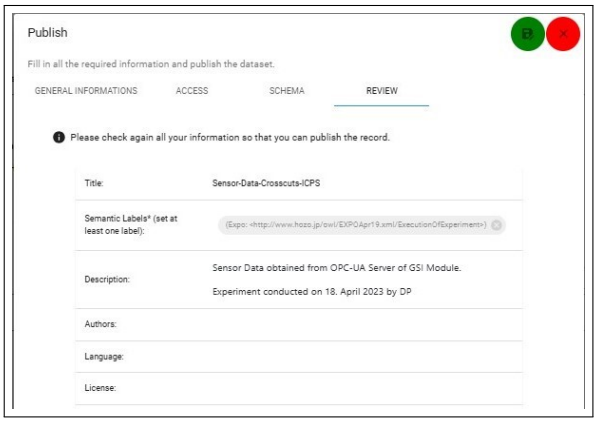
(a) Entering general metadata



(b) Configuring access rights



(c) Verifying schema extraction



(d) Review & publish to *Workspace*

Figure B.3: The four steps in the dialog of the *Ingestion* tab

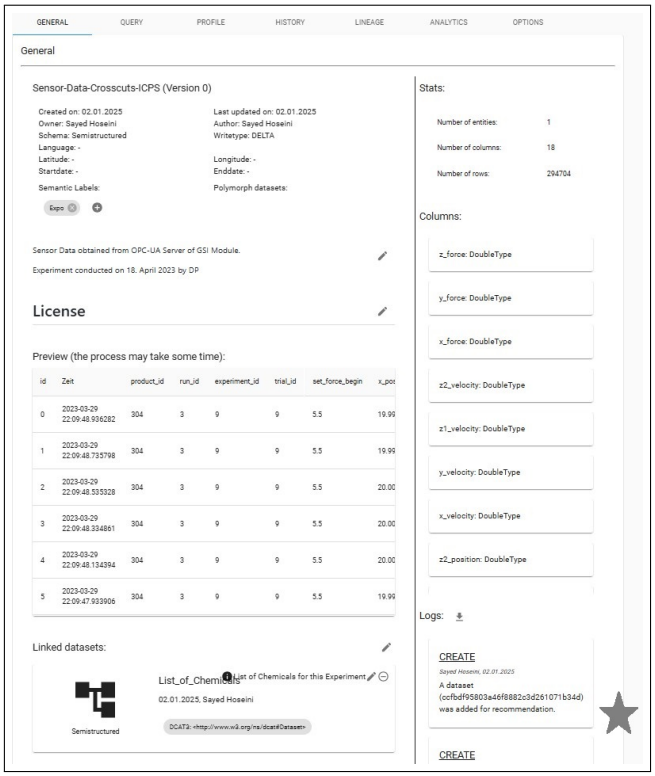


Figure B.4: *General* Tab of a *DataSet*'s view

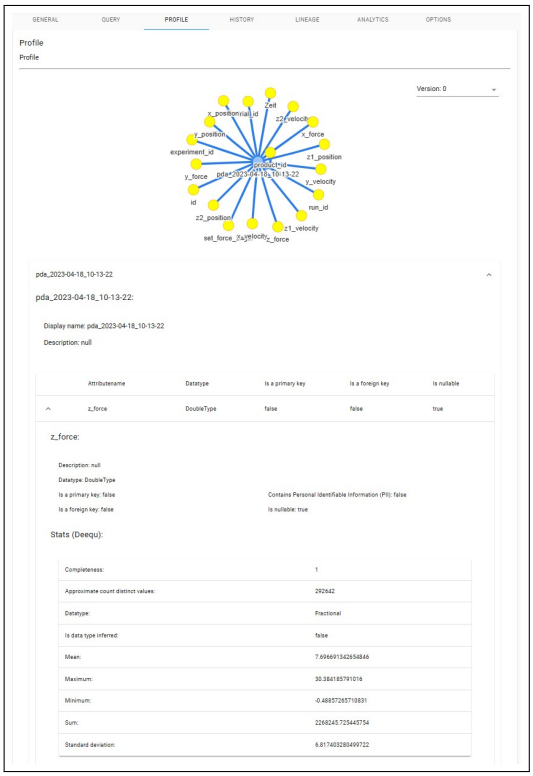
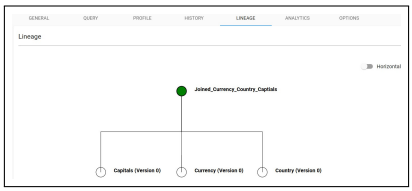
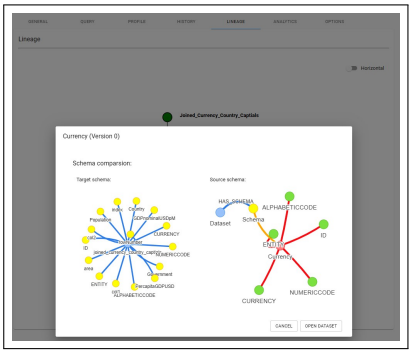


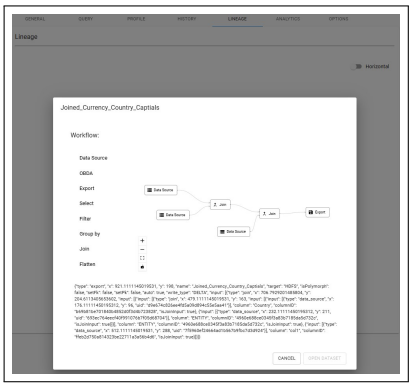
Figure B.5: *Profile* Tab of a *DataSet*'s view



(a) *Lineage* Tab overview

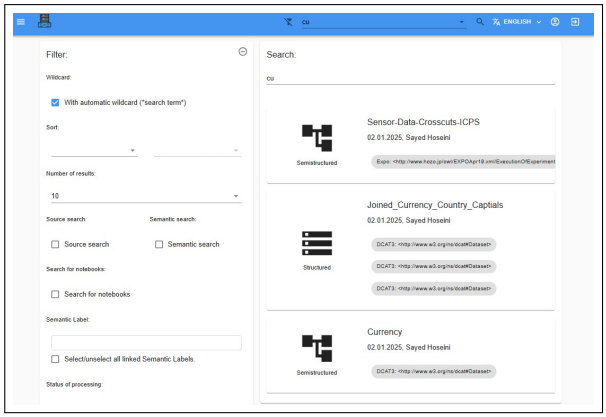


(b) Comparison of schemas

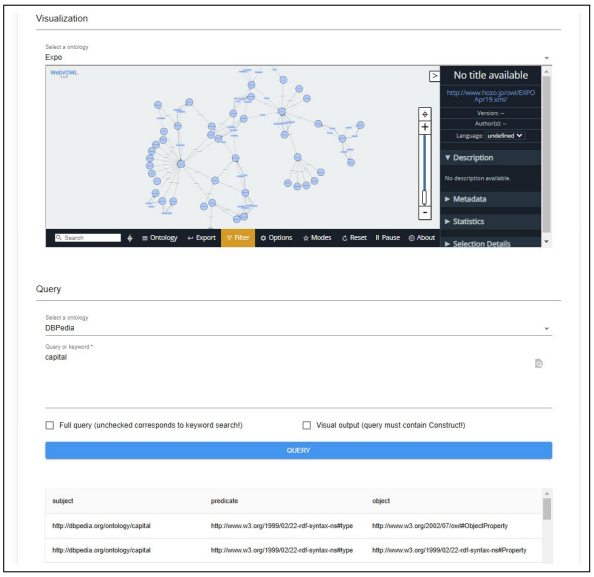


(c) Transformation workflow (see section 4.3.3)

Figure B.6: Illustrations of the *Lineage* Tab in SEDAR.

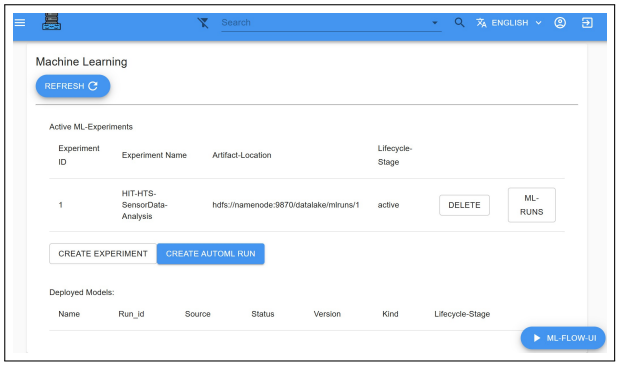


(a) Searching for *DataSets* in SEDAR

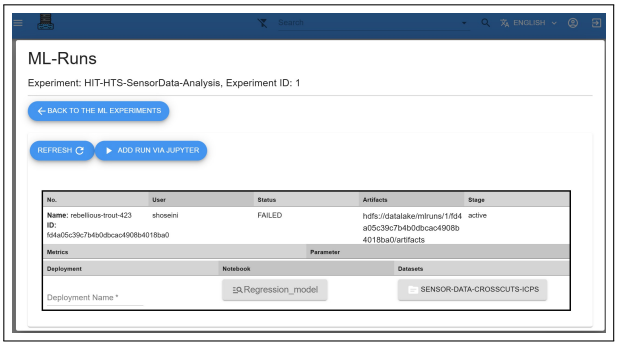


(b) Visualizing and querying ontologies within the Ontologies tab

Figure B.7: Search and ontology exploration functionalities in SEDAR.



(a) Listing experiments and deployed models



(b) View of runs associated to an ML experiment

Figure B.8: General view of the *ML* tab

C Semantic Data Management

C.1 IRI Reference Table

IRI identifier	Natural name
`wd:Q11424`	film
`wd:Q47703`	The Godfather
`wd:Q2526255`	film director
`wd:Q56094`	Francis Ford Coppola
`wd:Q5`	human
`wdt:P57`	director
`wdt:P1476`	title
`wdt:P31`	instance of

Table C.1: Natural-language glosses for the IRI identifiers used in the running example (Wikidata, RDF/RDFS/OWL/SKOS, XSD, and RML/R2RML).

C.2 Data Model for Semantic Labels

We illustrate the data model using two heterogeneous sources: (i) a CSV table describing a film, and (ii) a nested JSON document describing a person and their awards. The example highlights the distinct roles of records r and selectors k.

CSV s_{csv} contains the following row:

Title	Director	Length	Won_Oscar
Godfather	Coppola	2h 55min	1973

The source induces one record per row:

$$\mathcal{R}_{s_{\text{csv}}} = \{r_1^{\text{csv}}\}.$$

The set of keys is

$$\mathcal{K}_{s_{\text{csv}}} = \{\texttt{Title}, \texttt{Director}, \texttt{Length}, \texttt{Won_Oscar}\}.$$

The extraction function yields:

$$D_{s_{\text{csv}}}(r_1^{\text{csv}}) = \{(\texttt{Title}, \texttt{Godfather}),\\ (\texttt{Director}, \texttt{Coppola}),\\ (\texttt{Length}, \texttt{2h 55min}),\\ (\texttt{Won_Oscar}, 1973)\}.$$

This gives rise to the following instance-level data elements:

$$\{(s_{\text{csv}}, r_1^{\text{csv}}, \texttt{Title}, \texttt{Godfather}),\\ (s_{\text{csv}}, r_1^{\text{csv}}, \texttt{Director}, \texttt{Coppola}),\\ (s_{\text{csv}}, r_1^{\text{csv}}, \texttt{Length}, \texttt{2h 55min}),\\ (s_{\text{csv}}, r_1^{\text{csv}}, \texttt{Won_Oscar}, 1973)\}$$

Here, the record identifier r corresponds to a table row; positional information is not required in k because each key occurs at most once per record.

JSON s_{json} is given by:

```
{
  "name": "Coppola",
  "profession": "director",
  "awards": [
    { "award": "Oscar", "work": "Godfather", "type": "Best Film" },
    { "award": "Oscar", "work": "Godfather", "type": "Best Actor",
      "recipientName": "Marlon Brando" }
  ],
  "type": "per_project"
}
```

The document is treated as a single record:

$$\mathcal{R}_{s_{\text{json}}} = \{r_1^{\text{json}}\}.$$

Selectors are expressed as JSONPath expressions. Since the array `awards` is ordered and its elements differ structurally, positional information is included in the selectors:

$$\mathcal{K}_{s_{\text{json}}} = \{\texttt{\$.name}, \texttt{\$.profession}, \texttt{\$.type},\\ \texttt{\$.awards[0].award}, \texttt{\$.awards[0].work}, \texttt{\$.awards[0].type},\\ \texttt{\$.awards[1].award}, \texttt{\$.awards[1].work}, \texttt{\$.awards[1].type},\\ \texttt{\$.awards[1].recipientName}\}.$$

The extraction function yields:

$$D_{s_{\text{json}}}(r_1^{\text{json}}) = \{(\texttt{\$.name}, \texttt{Coppola}),$$
$$(\texttt{\$.profession}, \texttt{director}),$$
$$(\texttt{\$.type}, \texttt{per_project}),$$
$$(\texttt{\$.awards[0].award}, \texttt{Oscar}),$$
$$(\texttt{\$.awards[0].work}, \texttt{Godfather}),$$
$$(\texttt{\$.awards[0].type}, \texttt{Best Film}),$$
$$(\texttt{\$.awards[1].award}, \texttt{Oscar}),$$
$$(\texttt{\$.awards[1].work}, \texttt{Godfather}),$$
$$(\texttt{\$.awards[1].type}, \texttt{Best Actor}),$$
$$(\texttt{\$.awards[1].recipientName}, \texttt{Marlon Brando})\}.$$

The corresponding instance-level elements are:

$$\{(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.name}, \texttt{Coppola}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.profession}, \texttt{director}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.type}, \texttt{per_project}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[0].award}, \texttt{Oscar}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[0].work}, \texttt{Godfather}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[0].type}, \texttt{Best Film}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[1].award}, \texttt{Oscar}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[1].work}, \texttt{Godfather}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[1].type}, \texttt{Best Actor}),$$
$$(s_{\text{json}}, r_1^{\text{json}}, \texttt{\$.awards[1].recipientName}, \texttt{Marlon Brando})\}$$

C.3 RML Mapping for the running example

```
@prefix rr:  <http://www.w3.org/ns/r2rml#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql:  <http://semweb.mmlab.be/ns/ql#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wd:  <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix nv:  <http://example.org/nv/> .
@prefix ex:  <http://example.org/base/> .

##############################
# Movies (CSV)
##############################
ex:MovieTM
  a rr:TriplesMap ;
  rml:logicalSource [
    rml:source "movies.csv" ;
    rml:referenceFormulation ql:CSV ;
  ] ;
```

```
19
20    rr:subjectMap [
21      rr:template "http://example.org/base/movie/{Title}" ;
22    ] ;
23
24    rr:predicateObjectMap [
25      rr:predicate wdt:P31 ;
26      rr:objectMap [ rr:constant wd:Q11424 ] ;    # film
27    ] ;
28
29    rr:predicateObjectMap [
30      rr:predicate wdt:P1476 ;
31      rr:objectMap [ rml:reference "Title" ; rr:language "en" ] ;
32    ] ;
33
34    rr:predicateObjectMap [
35      rr:predicate wdt:P57 ;
36      rr:objectMap [
37        rr:parentTriplesMap ex:PersonTM ;
38        rr:joinCondition [
39          rr:child  "Director" ;
40          rr:parent "name" ;
41        ] ;
42      ] ;
43    ] ;
44
45    rr:predicateObjectMap [
46      rr:predicate nv:wonOscarYear ;
47      rr:objectMap [ rml:reference "Won_Oscar" ; rr:datatype xsd:gYear ] ;
48    ] .
49
50  ##############################
51  # Person (JSON root)
52  ##############################
53  ex:PersonTM
54    a rr:TriplesMap ;
55    rml:logicalSource [
56      rml:source "person.json" ;
57      rml:referenceFormulation ql:JSONPath ;
58      rml:iterator "$" ;
59    ] ;
60
61    rr:subjectMap [
62      rr:template "http://example.org/base/person/{name}" ;
63    ] ;
64
65    rr:predicateObjectMap [
66      rr:predicate wdt:P31 ;
67      rr:objectMap [ rr:constant wd:Q5 ] ;        # human
68    ] .
69
70  ##############################
71  # Oscar categories from JSON awards[*] attached to the movie
72  # (adds: Best Film + Best Actor for Godfather)
73  ##############################
74  ex:MovieOscarFactsTM
75    a rr:TriplesMap ;
76    rml:logicalSource [
77      rml:source "person.json" ;
78      rml:referenceFormulation ql:JSONPath ;
79      rml:iterator "$.awards[*]" ;
80    ] ;
81
82    rr:subjectMap [
83      rr:template "http://example.org/base/movie/{work}" ;
84    ] ;
85
86    rr:predicateObjectMap [
87      rr:predicate nv:wonOscarCategory ;
88      rr:objectMap [ rml:reference "type" ] ;
89    ] ;
90
```

```
  rr:predicateObjectMap [
    rr:predicate nv:wonOscarAward ;
    rr:objectMap [ rml:reference "award" ] ;
  ] ;

  # only present for the Best Actor entry in your JSON
  rr:predicateObjectMap [
    rr:predicate nv:wonOscarRecipientName ;
    rr:objectMap [ rml:reference "recipientName" ] ;
  ] .
```

Listing C.1: RML Mapping for the running example.

C.4 Semantic Modeling in SEDAR

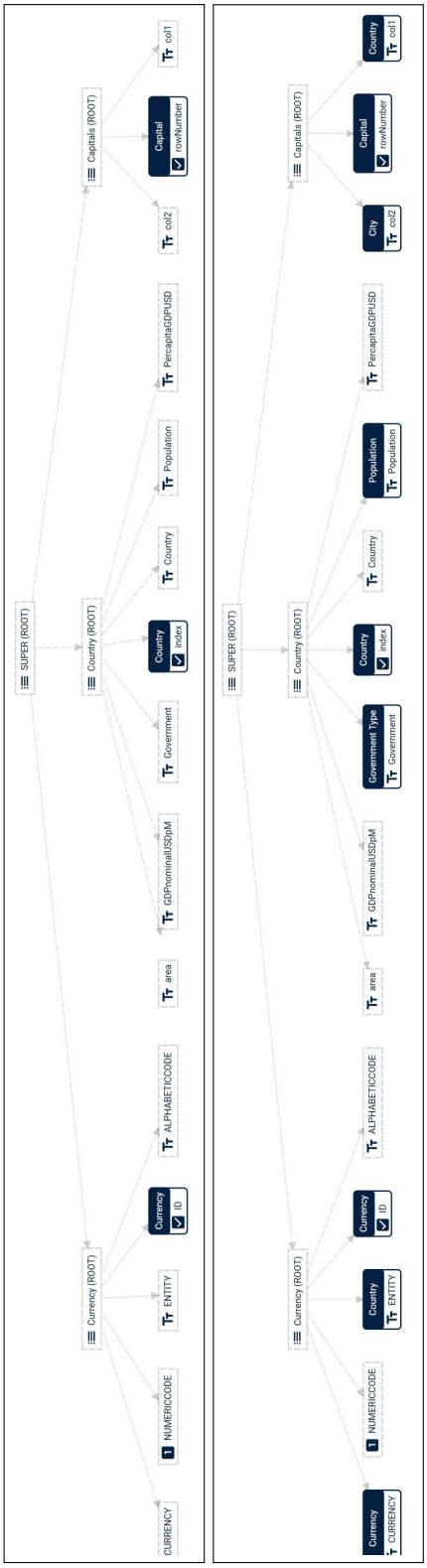


Figure C.1: Combined semantic model with initial annotations on the primary keys (top) and finalized semantic model contextualizing three datasets (bottom).

D KG-enhanced DS-Agent

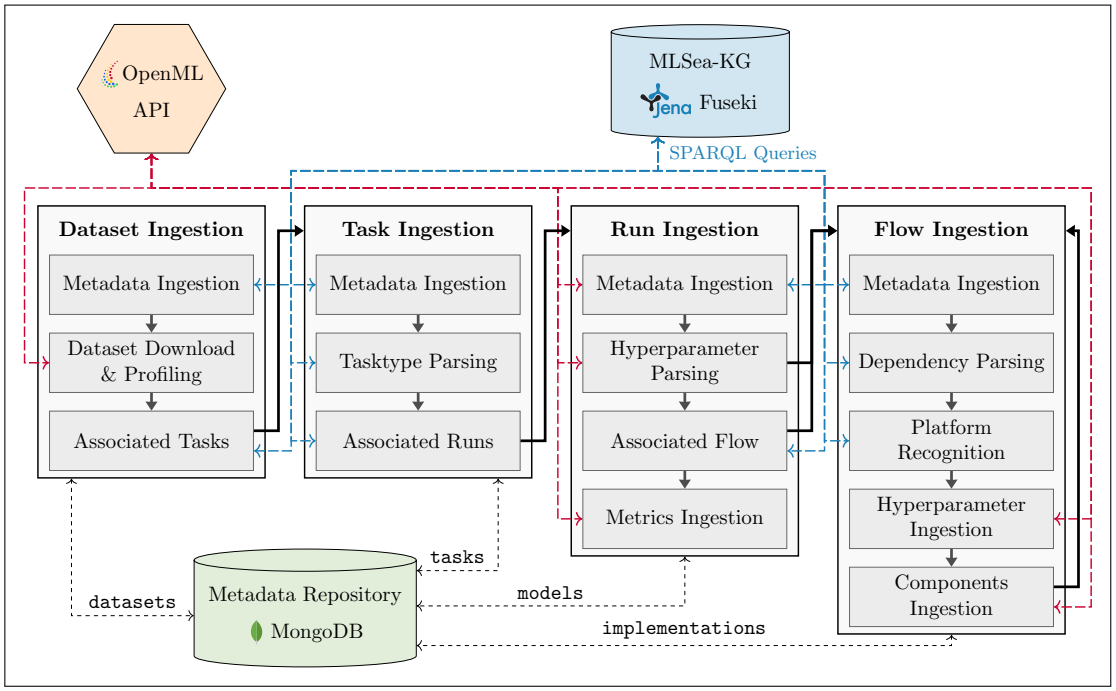


Figure D.1: Proposed ingestion pipeline

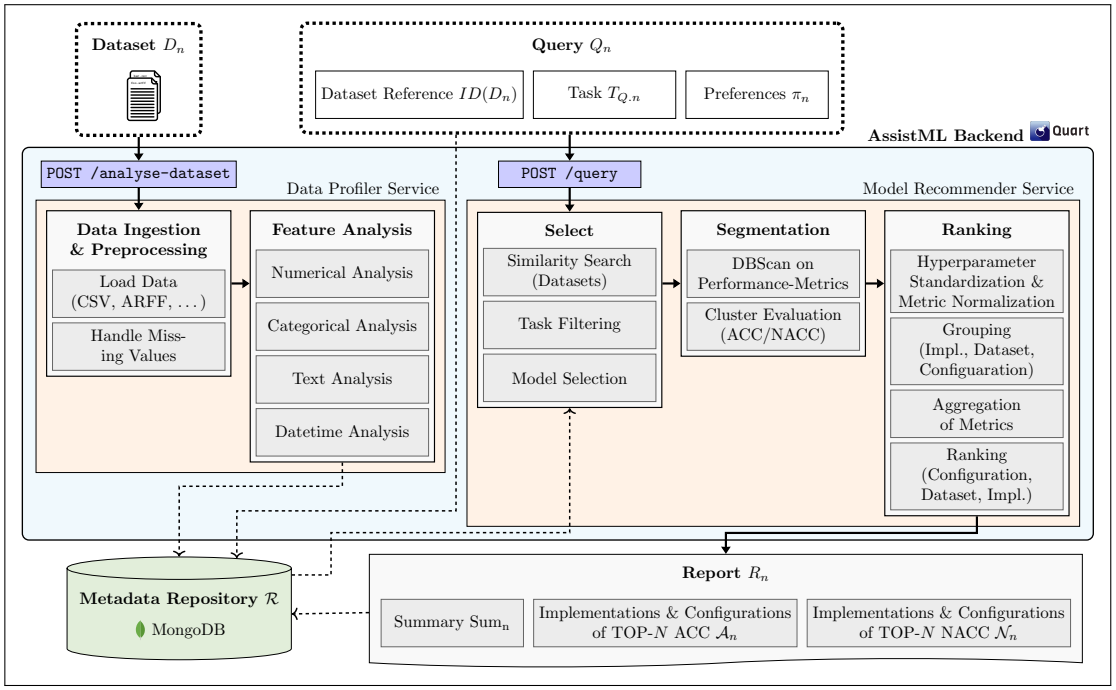


Figure D.2: Proposed backend for generating reports

E Prompts

Manager System Prompt

```
You are the Manager Agent. You are responsible for managing the
overall workflow, coordinating the other agents (CODE, TOOL).
```

Manager Prompt Template

```
You are a workflow coordinator responsible for managing agents and
    selecting the next best action to fulfill the user query. A
    Workspace is always available by default. You can choose between
    TOOL (tool calling) or CODE (custom code).

Current User Query: <USER_QUERY>
These objects are available for tool calling in cache: <TOOL_OBJECTS>
These are the other objects in the cache: <OTHER_OBJECTS>
Here are some classes and methods for context:
    <AVAILABLE_CLASSES_AND_METHODS_OBTAINED_FROM_RETRIEVAL>

Now, to answer the current user query, choose the next best action:
TOOL: If there is a method available that can be used to answer the
query. And if there is an instance for tool calling that has this
method.
CODE: If you need to generate custom code to answer the query. For this,
    all objects in cache can be used. Also methods and attributes of the
    objects. Also use CODE to access attributes of the objects in cache.

Output the next action in the following format:
{  "action": "<action>", "tool_object": "<tool_object> or NONE" }
You only need to provide the tool_object when using the TOOL action.
    When using CODE, set tool_object to NONE.
=====================================================================
Examples: <FEW_SHOT_EXAMPLES>
=====================================================================
Determine the next step in the workflow for the current user query:
<USER_QUERY>
<LAST_OUTPUT>
and output ONLY JSON in the specified format. Use TOOL if there is an
    object in cache and a suitable method. Use CODE if you need to
    generate custom code.
```

Manager Few-Shot Example

```
"Update the current user's name to 'Alice'."

Tool objects in cache (containing current user):
{ "_USER_3n1k5jld": User... }

Available classes and methods:
User:
  name (str): The name of the user.
  email (str): The email address of the user.
  def update(self, new_email: str = None, firstname: str = None,
             lastname: str = None, username: str = None,
             is_admin: bool = None) -> User:
  def delete(self):
    ...
Output:
{  "action": "TOOL", "tool_object": "_USER_3n1k5jld" }
```

E.1 Semantic variation in the evaluation dataset

Examples of semantic variation in the evaluation dataset for the concrete task of uploading a dataset.

- Create a dataset from the file 'Capitals.json'. Then ingest it.
- Ingest the file 'Capitals.json' into the appropriate storage system and activate versioning.
- Build a dataset utilizing 'Capitals.json' and follow it with an ingestion step.
- Utilize 'Capitals.json' to build a dataset, and then proceed with the ingestion process.
- Use 'Capitals.json' to assemble a dataset and then carry out the ingestion.

The final dataset consists of 135 queries (27 base + 108 filtered variations) with an average similarity score of 0.7822 computed by the Sentence Transformer model. Table E.1 summarizes the similarity scores across query types, demonstrating the variety in phrasing and coverage of use cases.

Table E.1: Average cosine similarity scores for different query types each extended by four semantically distinct rephrasings

Query Type	**Similarity Score**
Create And Ingest	0.7760
Find Related Datasets	0.8356
Get all datasets	0.5282
Get All Versions	0.6753
Get Dataset Description	0.8246
Get Lineage	0.7933
Get Linked Datasets	0.7839
Get Row Count	0.8418
Get Stats	0.6148
Labeling and Semantic Mapping	0.8205
List Columns	0.8257
Create Binary Classification	0.8076
List Tags	0.7927
List Users	0.6578
Query Semantic Mapping	0.8131
Search Chemical Datasets	0.7976
Update Dataset Title	0.8335
Upload Ingest Two	0.8118
Versions And Deltas	0.6933
Workspace Title	0.8284
Create Dataset	0.8311
Create Dataset And Find Tag	0.8254
Create Ontology	0.8155
Create Regression	0.8397
Create semantic model	0.8275
Delete Dataset	0.8009
Describe Dataset	0.8244
Global Average	**0.7822**

F End-To-End ML with LLMs and Semantic Data Management

```
import numpy as np
import pandas as pd
df, df_viskos, df_abrasion = pd.read_excel('./RunStatistic.xlsx', skiprows=8),
    pd.read_csv('./viskos_means.csv', delimiter=";"),
    pd.read_csv('./abrasionstests.csv', delimiter=";")
columns = ["Run_ID","Experiment", "Basis", "Matt1", "Matt2", "Wachs", "Verdicker",
    "t_To_Mix", "t_To_Dispense",
          "thickness", "gloss_60", "gloss_85", "haze", # README!!! offset, not all
              useful
          "CIELAB_10deg_L", "CIELAB_10deg_a", "CIELAB_10deg_b"] # README !!! not all
              useful
df = pd.DataFrame(columns=columns)
for run_id in df_tr["Run ID"].unique():
    df_temp_run = df_tr[df_tr["Run ID"] == run_id]
    for exp_id in df_temp_run["Experiment"].unique():
        df_temp= df_temp_run[df_temp_run["Experiment"] == exp_id]
        # extract dispensed amounts
        v1 = df_temp[df_temp['Dispense Material'] == 'Basismodul']['Dispensed
            Amount'].iloc[0]
        v2 = df_temp[df_temp['Dispense Material'] == 'Mattmodul 1']['Dispensed
            Amount'].iloc[0]
        v3 = df_temp[df_temp['Dispense Material'] == 'Mattmodul 2']['Dispensed
            Amount'].iloc[0]
        v4 = df_temp[df_temp['Dispense Material'] == 'Wax HM 7-004']['Dispensed
            Amount'].iloc[0]
        v5 = df_temp[df_temp['Dispense Material'] == 'Thickener HZ 3-88']['Dispensed
            Amount'].iloc[0]
        # Calculate times
        t_dispense = pd.to_datetime(df_temp[df_temp['Node'] =='3580: Dispense Liquid
            - M']['UTC'].iloc[0])
        t_mix = pd.to_datetime(df_temp[df_temp['Node'] == '3583: Mix']['UTC'].iloc[0])
        t_drawdown = pd.to_datetime(df_temp[df_temp['Node'] == '3585: Draw
            Down']['UTC'].iloc[0])
        t_to_mix, t_to_drawdown = t_mix - t_dispense, t_drawdown - t_mix
        # get the characterization results
        match = re.search(r'\(#(\d+)\)', df_temp['Experiment'].iloc[0])
        n = match.group(1) if match else None
        m_item = next((i for i in os.listdir('measurements') if n in i), None)
        c = os.listdir(f'measurements/{m_item}') if m_item else []
        ### thickness | Only third measurement, Offset of 10
        df_thickness = pd.read_csv('DoE1/'+m_item+"/"+next((i for i in c if
            "minitest" in item), None))
        thickness = df_thickness.iloc[2][0] - 10
        ### Gloss / Haze | Messpunkt 2 | GZ: Gloss60, 85, Haze, Dritte Zeile (Zweiter
            Messwert)
        df_h_g = pd.read_csv('DoE1/'+m_item+"/"+next((i for i in c if "haze" in
            item), None)).iloc[:, 1:]
        gloss_60, gloss_85, haze = df_h_g.iloc[1][1], df_h_g.iloc[1][2],
            df_h_g.iloc[1][3]
        ### Spectrum | Messpunkte 2+3 | GZ: nur Zeile 3 nehmen (zweiter Messwert)
        df_h_g = pd.read_csv('DoE1/'+m_item+"/"+next((i for i in c if "xrite" in
            item), None)).iloc[:, 1:]
        s_1, s_2, s_3 = df_h_g["CIELAB_L"].iloc[1], df_h_g["CIELAB_a"].iloc[1],
            df_h_g["CIELAB_b"].iloc[1]
        new_row = pd.DataFrame({"Run_ID": run_id,"Experiment_ID": exp_id,
                                "Basis": v1,"Matt1": v2,"Matt2": v3,"Wachs":
                                    v4,"Verdicker": v5,
                                "t_To_Mix": t_to_mix, "t_To_Dispense": t_to_drawdown,
                                "thickness": thickness,"gloss_60":
                                    gloss_60,"gloss_85": gloss_85,"haze": haze,
                                "CIELAB_10deg_L":s_1 -
                                    4.36,"CIELAB_10deg_a":s_2,"CIELAB_10deg_b":s_3})
        df = pd.concat([df, new_row], ignore_index=True)
df = pd.concat([df, df_viskos]) # Viscosity
df['Run_ID'], df['Experiment_ID'] = df['Run_ID'].astype(int),
    df['Experiment_ID'].astype(int)
df = pd.merge(df, df_abrasion, on=['Run_ID', 'Experiment_ID'], how='inner')
```

Listing F.1: Target code for wrangling the data set in Python

```
import sklearn
train_columns =
    ['Matt1','Matt2','Wachs','Verdicker','t_To_Mix','t_To_Dispense','thickness',
        'haze','C_10deg_L']
target_columns = [ 'gloss_85',  'Viskos_500', 'abrasion2']
X_train, y_train = np.array(df[train_columns]), np.array(df[target_columns])
X_test, y_test = np.array(df_test[train_columns]), np.array(df_test[target_columns])
scaler, scaler_y = MinMaxScaler((0, 1)), MinMaxScaler((0, 1))
X_train_scaled, y_train_scaled = scaler.fit_transform(X_train),
    scaler_y.fit_transform(y_train)
X_test_scaled, y_test_scaled = scaler.transform(X_test), scaler_y.transform(y_test)
# Hyperparameters
nu_values, length_scale_values, noise_level_values = [1.5, 0.5],  [0.5 * 1e-2, 2.0 *
    1e-2], [0.01, 0.1]
hyperparameter_combinations = list(product(nu_values, length_scale_values,
    noise_level_values))
kernels = [RBF(), Matern(),  WhiteKernel(), ExpSineSquared(),
    RationalQuadratic(),(RBF() + WhiteKernel()),
(Matern() + WhiteKernel()),  (ExpSineSquared() + WhiteKernel()),
    (RationalQuadratic() + WhiteKernel())]
for nu, length_scale, noise_level in hyperparameter_combinations:
    kernels.append(Matern(nu=nu) + WhiteKernel(noise_level=noise_level) +
        RBF(length_scale=length_scale))
gpr_param_grid = {'kernel': kernels, 'n_restarts_optimizer': [5, 10, 15]}
# Fit the GPR grid search to the training data
multi_gpr = MultiOutputRegressor(GaussianProcessRegressor())
gpr_grid_search = GridSearchCV(multi_gpr, gpr_param_grid, cv=5, n_jobs=-1,
    scoring='neg_mean_squared_error')
gpr_grid_search.fit(X_train_scaled, y_train_scaled)
y_pred_gpr = gpr_grid_search.best_estimato.predict(X_test_scaled)
mse_gpr, mae_gpr, mape_gpr = mean_squared_error(y_test_scaled, y_pred_gpr),
    mean_absolute_error(y_test_scaled, y_pred_gpr),
    mean_absolute_percentage_error(y_test, y_pred_gpr)
print("MSE:", mse_gpr, "MAE:", mae_gpr, "MAPE:", mape_gpr)
```

Listing F.2: Crafting the ML model, imports truncated