SoftwareX 23 (2023) 101487

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ProLoaF: Probabilistic load forecasting for power systems

Gonca Gürses-Tran [a,*], Florian Oppermann [b], Antonello Monti [a,b]

[a] Fraunhofer-Institut für Angewandte Informationstechnik FIT, Hüttenstr. 5, Aachen, 52068, Germany
[b] Institute for Automation of Complex Power Systems, RWTH Aachen University, Mathieustr. 10, Aachen, 52072, Germany

ARTICLE INFO

Article history:
Received 4 October 2021
Received in revised form 1 February 2023
Accepted 24 July 2023

Keywords:
Machine Learning
MLOps
PyTorch
Python
Short-term load forecasting

ABSTRACT

Today, the energy supply does not follow the demand in a controlled manner anymore. Thus, forecasting the electricity consumption became essential for the operation of power systems. Already numerous open source software tools exist that provide forecasting models, which are configurable for different forecasting tasks. In the case of electrical energy demand, a change in the geographical or temporal settings, requires specific domain knowledge on relevant data and influencing factors that are to be considered when developing data-driven forecasting models. With *ProLoaF*, we propose a holistic machine-learning based forecasting project, which offers the developer a continuous deployment of reliable forecasts for the power system domain. ProLoaF serves for probabilistic forecasts of the electric energy consumption and non-controllable generation in future power system operation. By overlapping Machine Learning (ML), DevOps and power systems engineering disciplines, we aim to accelerate future forecasting model development by reducing consultation work between domain experts.

Code metadata

Current code version	v0.1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00190
Code Ocean compute capsule	https://codeocean.com/capsule/5396363/
Legal Code License	Apache License, Version 2.0
Code versioning system used	git
Software code languages, tools, and services used	Python, Jupyter Notebook, Docker
Compilation requirements, operating environments & dependencies	Python $\geq$ 3.6
Link to developer documentation/manual	public/automation/plf/proloaf
Support email for questions	sogno-tsc+help@lists.lfenergy.org

Software metadata

Current software version	v0.1.5
Permanent link to executables of this version	github.com/sogno-platform/proloaf
Legal Software License	Apache License 2.0
Computing platforms/Operating Systems	Linux, Windows
Installation requirements & dependencies	Python $\geq$ 3.6
If available, link to user manual - if formally published include a reference to the publication in the reference list	public/automation/plf/proloaf
Support email for questions	sogno-tsc+help@lists.lfenergy.org

1. Motivation and significance

Growing shares of renewable energy sources have led to an increasing need for short-term time series forecasts of generation and load that also quantify the influence of, e.g., meteorological

* Corresponding author.
E-mail address: gonca.guerses-tran@fit.fraunhofer.de (Gonca Gürses-Tran).
URL: https://www.digitale-energie.fraunhofer.de (Gonca Gürses-Tran).

https://doi.org/10.1016/j.softx.2023.101487

uncertainties. Although numerous open source packages exist to generate forecasting models with low development effort, in the case of electrical energy demand, each new geographical location poses a new challenge on the specific influencing factors that need to be considered. Thus, forecasting model developers are required to apply domain knowledge to choose from a large set of available methods, to adapt and finally to tune the chosen software.

With *ProLoaF* [1], we introduce a machine learning (ML) based forecasting project in one of the most commonly used programming languages, `Python`, to significantly accelerate forecasting model development in the energy sector. The chosen ML framework `PyTorch`[2] allows the developer to build customised forecasting models and accelerate the training process with the use of graphics processing units. In the basic scenario, *ProLoaF* requires the historic time series of the prediction target. Further explanatory time series data can be added to improve the performance upon availability of, e.g., energy generation profiles, meteorological profiles, and calendric information, specific to the geographical location under study. The most relevant prediction targets in power system operation are electric power generation, power consumption, and electricity prices. Initially, we targeted for short-term power consumption forecasts, better known as short-term load forecasts (STLF). Concerning short-term load forecasts, a recent literature review [3] divides the vast number of contributions in the area of STLF under uncertainty into *methodological contributions*, which address improved forecasting algorithms, and *applied contributions*, which address the proposed algorithms solely as a vehicle for decision-making in business processes. The work in [4] and [5] indicate that there is no single model type, which qualifies best for all forecasting tasks. Thus, forecasting competitions, such as the M forecasting series, are useful to identify the most promising applications for time series applications. Among others, [4,6] and [7] list sequence-to-sequence learning, extreme gradient boosting trees, and exponential smoothing techniques, as most recent and performant model candidates. However, only few published forecasting projects exist, which contribute to the scientific and industrial community with accompanying open-source software that is made available for the purpose of faster adaption of recent advances in forecast methodologies. In addition, only a considerably small subset of software products apply a holistic framework for the forecasting tasks in the energy domain. Both requirements, the replicability and the holistic approach, are relevant to facilitate further integrated forecasting model developments in the context of power system services.

1.1. Software properties

Table 1 summarises the properties of open source forecasting tools, which match these requirements, and thus, facilitate a fast development of STLF. All contributions are applicable to forecasting model development and relieve researchers and developers from software engineering tasks. Concerning the baselines, we choose naïve and statistics based forecasting algorithms as described in [8] are readily available both in the `R` forecasting library [9] and in Python `statsmodels` [10]. We integrate these baseline models, to be able to benchmark the forecast performance of more advanced approaches. In *ProLoaF*, we integrate seasonal/decomposed persistence models, exponential smoothing, `auto-arima` [11], and `GARCH` [12] as baseline forecasting tools and produce forecasts on a rolling horizon window over the test data set. By re-fitting the models on a sliding-window basis over the data, we aim to improve the forecast accuracy that can be achieved with the baseline models. While *ProbCast* [13] and *Prophet* [14] propose Time Series Regression and Time Series Decomposition respectively as advanced forecasting methods, *ProLoaF*, *GluonTS* [15], *PyTorch Forecasting* [2] and *Neural Prophet* [16] apply deep learning approaches. In particular, *GluonTS* and *PyTorch Forecasting* primarily work as wrappers that bundle existing models, tools, and components for time series applications.

Key properties to accelerate the development of forecasting models are the analysis of performance indicators during the training process, and the diagnostic tools, such as visualisation and quantification of the errors on out-of-sample predictions. Especially when considering the forecasting tool in a production environment, i.e., as a decision support for power system engineers, the automation of training, prediction and diagnostics steps is required. Interface developments can further facilitate the embedding of training and prediction steps into target systems so that roles of the forecast user and the forecast developer can be separated. Notice that with the separation of the roles of developer and user, the requirements for the explainability of the forecast model increase. This is due to the fact that the forecast user has no insights on how and based on which inputs the underlying model achieves the prediction. Statistical methods, such as regression methods, by their very nature, allow the user more explainability than deep neural networks due to the straightforward relationship between model inputs and predictions. As the power system domain involves critical infrastructure, it is vital that forecasters can have confidence in their predictions, through explainability. For this, *ProLoaF* provides model-agnostic tools to visualise the most salient input features per time step and thereby make the underlying ML model explainable to some extent.

1.2. Experimental setting

The proposed software package can be used as a back-end element of an existing software stack, e.g., in forecasting pipelines for deployment in power system control rooms. Moreover, it contains *Jupyter notebooks* and Python scripts dedicated to serve as high-level user interfaces. These interfaces are aimed to facilitate forecast development with *ProLoaF*, in the sense of an advanced user guide. In the following of this paper, we refer to these high-level interfaces as *entry points*. The notebooks, convey a deeper understanding of the underlying evaluation criteria that ProLoaF uses both for the training and diagnostics steps. The code examples are all built upon open power system data [17].

Forecasts that are generated with *ProLoaF* may not always result in good forecasts. This can be the case if the training data is not sufficient or extreme situations occur, e.g., planned or forced outages. Such situations, which would likely lead to inaccurate forecasts, are rarely present in the training data. Thus, the current version of ProLoaF requires a certain degree of judgement or prior definition of qualitative indicators by the human forecast developer and user to judge on the reliability of the forecast in the considered operational task.

2. Software description

2.1. Software architecture

We express the main software functionalities with four Python scripts that serve as entry points for *ProLoaF*, as depicted in Fig. 1. The entry points correspond to distinct steps in a forecasting pipeline, described in Section 2.2. All of these entry points are independent of each other but make use of a common code base consisting of handler modules for data, configuration files, baseline algorithms, and the forecasting model, a custom data container (`utils.tensorloader`) and metrics for evaluation. Fig. 1

Table 1
Qualitative comparison with existing tools.

Integrated properties	ProLoaF	ProbCast	GluonTS	PyTorch-forecasting	Prophet	Neural prophet
Version	v0.1.0	v0.1 beta	v0.8.0	v0.9.0	v1.0	v0.2.7 beta
Language/Framework	PyTorch	R	PyTorch	PyTorch	PyTorch, R	PyTorch
Baseline Models	●	-	●	◐	●	-
Forecasting Technique	RNN	Regression	Various	Various	Decompos.	AR-NN
Prediction Intervals	●	●	●	●	●	-
Paral. Hyperp. Tuning	●	-	n.d.	●	●	●
Model Exploration	●	-	●	-	●	-
Diagnostics Tools	●	●	●	●	●	●
Degree of Automation	◐	-	●	◐	●	-
Explainability	◐	●	n.d.	◐	●	◐
Applicable to STLF	●	●	◐	◐	◐	◐
Applied on STLF	●	●	-	-	-	◐

● = provides property; ◐ = partially provides property; - = does not provide property.

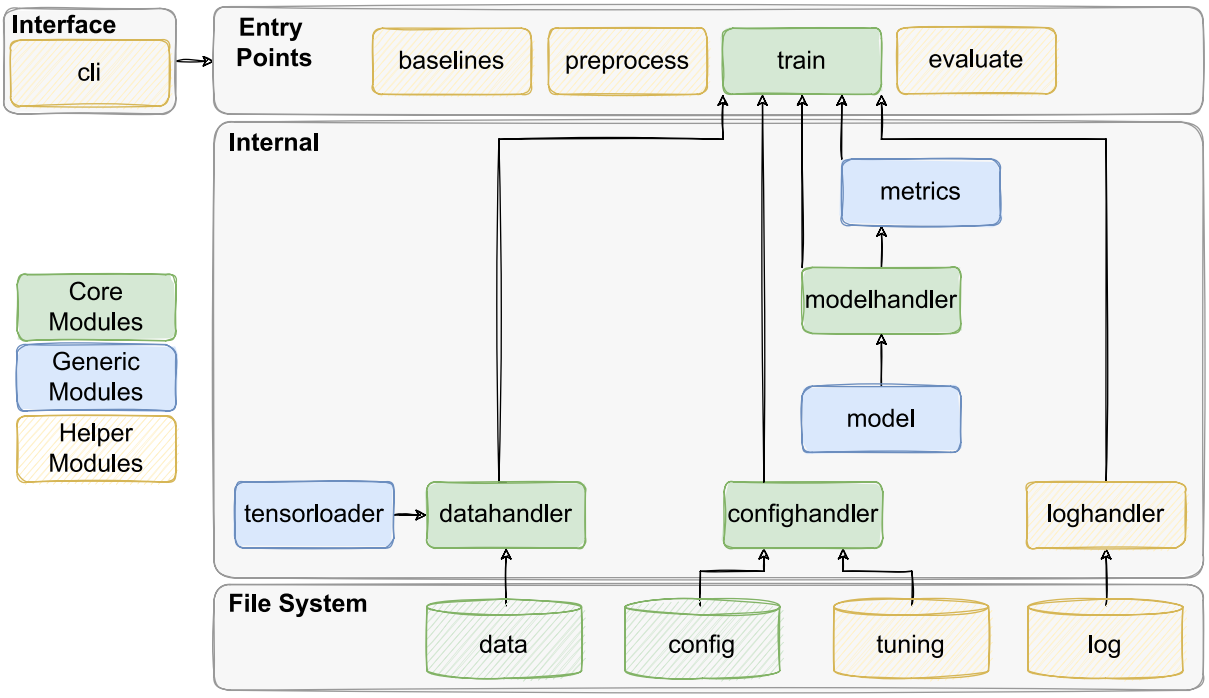

Fig. 1. Dependency graph of ProloaF train module..

particularly depicts the dependency of the training algorithm from the generic *ProLoaF* forecast modules. In total, four main entry points, i.e. pre-processing, training, evaluation (diagnostics) and benchmarking exist with different dependencies, further described in [1]. The handlers provide functionalities for all tasks in the context of forecasting, e.g., with the `utils.modelhandler` we can train a forecasting model, save it, and load it from the file system again to generate predictions.

All the handlers can be used independently of the entry points and from each other. In the same way, the forecasting model and metrics can be used independently of their respective handler.

ProLoaF is highly configurable without the need for familiarity with the source code. This is achieved by means of a configuration file in JavaScript Object Notation (JSON). The format can represent relatively complex data structures while still being fairly readable. It is thus well suited to configure the input data handling for feature engineering, the training, and the hyperparameter selection for tuning. Using a single configuration file, moreover, avoids misalignment within one defined forecasting job. Such a misalignment could occur, for instance, if the training was conducted with a certain set of features and later on the prediction is attempted with a different set of features, which would eventually lead to poor results or already fail during execution.

2.2. Software functionalities

As described in Section 2.1, *ProLoaF*'s functionalities are independently usable but the most typical user scenario is to train an ML based model and deploy the forecasts in production. The following steps contain a summary of *ProLoaf*'s functionalities:

1. The input features and configuration are imported from the file system.
2. Optionally, the data is pre-processed.
3. A minimal feature engineering step is applied, with the most relevant time series features in power systems.
4. The collected data is transformed into the data format needed by the sequence-to-sequence model.
5. A sequence-to-sequence model is constructed, trained and validated.
6. Forecast performance metrics are computed on out-of-sample predictions with unseen data.
7. The resulting performance metrics are returned and displayed for back-testing purposes.
8. We track an increase or decline in the forecast performance, by logging the resulting configuration and score of

previous stages, which guarantees efficiency through the mitigation of repetitive re-training needs.

In summary, *ProLoaF* assembles all steps for an MLOps project, i.e., the application of DevOps principles to Machine Learning applications [18], because of the way it uses overlapping ML, DevOps and power systems engineering disciplines. The rest of this chapter, elaborates on some of these functionalities.

2.2.1. Data pre-processing

Historical data, required for training, is often quite inhomogeneously recorded and spread over multiple files, usually in a tabular format. In addition, this data usually contains gaps or invalid values which cannot be interpreted by the forecasting model. For this reason, *ProloaF* provides an entry point that takes a description of the disorganised data and merges it accordingly. It normalises date/time formats and time zones, stacks files appropriately, narrows the data down to the common starting and end times, and interpolates missing values in the data. In addition, a one-hot encoding of time properties is added in this step, indicating weekdays and the hour of the day. This results in a normalised data representation that includes all the relevant data in a single `.CSV` file and which is guaranteed to follow the formatting conventions used during the training and forecasting process. As mentioned in Section 2.1, this pre-processing is independent of the rest of the package and can thus easily be used to normalise data for other projects or be replaced/skipped if historical data is available in a different custom format. During training, the data is split into training, validation and test sets and then scaled according to the training set. Available scalers are the `StandardScaler`, `RobustScaler`, and `MinMaxScaler` provided by scikit-learn [19]. These can be specified easily using the configuration file. This is done during training and not pre-processing since the scaling parameters are part of learned parameters and thus belong to the forecasting model.

2.2.2. Feature engineering

One of the most important aspects in the development of an ML model is the selection of the input data. The default exogenous features used in *ProLoaF*'s examples are local meteorological information (if available), and one-hot and trigonometric encodings of time series properties, namely hours, months and weekdays, which performs well for most STLF tasks. For a more advanced feature engineering approach to time series problems, we recommend the developer take a closer look at features that can be extracted from the underlying time series, as provided by, e.g., the TSFEL library [20].

2.2.3. Forecast model

In the current version of *ProLoaF*, the generic module `model.py` contains an encoder–decoder model class that builds upon PyTorch's basic building blocks `torch.nn`. The member variables of the encoder–decoder class include the data scaler parameters, to apply equally fitted scalers on the input data during training and testing. Using keyword arguments that are fetched from the configuration file, the sequence-to-sequence model core uses, e.g. LSTM or GRU implementations. The list of arguments can further contain dropout rates, number of layers, and the number of input features per encoder/decoder neural network. Previous work [21] provides a practical description and application of this forecasting model with a detailed specification of exemplary time series features and shows that it has outperformed earlier developed forecasting methods [22].

2.2.4. Hyperparameter tuning and logging

To find the optimal hyperparameter setting, *ProLoaF*'s train function builds upon the define-by-run hyperparameter tuning framework *Optuna* [23]. *Optuna* searches through randomised hyperparameters with a shared memory over multiple processing cores and adapts the search space with each training step. Both this parallelisation and the early pruning of inadequate parameterisations accelerate the search for suitable hyperparameters significantly compared to a randomly sampled search [19].

To use the hyperparameter exploration option, the exploration parameter has to be set to 'true' in the configuration. Furthermore, a tuning configuration has to be specified with the hyperparameters of interest, their respective probability distribution and a pre-defined stopping criterion, such as the number of trials or a timeout limitation. If there are no existing records of a better performing model for this target in earlier logs, the initial configuration file is overwritten with the newly found hyperparameters that result in the lowest loss on the validation set. In addition to the determined hyperparameters, the total computation time and training/validation loss are also logged continuously in the active development path (`./logs`).

2.2.5. Model exploration

Developers of forecasting models may find a visualisation and tooling kit useful for targeted ML experimentation. Since PyTorch supports TensorFlow's *TensorBoard* development [24], the training procedure in *ProLoaF* makes use of *TensorBoard*'s loss-tracking, network graph, bias histograms and parallel hyperparameter plots. This facilitates the training process exploration and tracking. The dedicated files are stored in the active developer's path (`./runs`) and can be explored during and after the training step.

2.2.6. Diagnostics tools

To train and evaluate the forecasting model, deterministic and probabilistic metrics are implemented. `metrics.py` serves as a library of scoring techniques for training but is also used to evaluate the predictions compared to the true values, allowing the study of further probabilistic metrics that are distinguishable from the previously chosen loss function. In this way, any bias in the performance benchmarks is omitted. In addition, the developer can easily switch between parametric and non-parametric evaluation functions, depending on the underlying data that the model should fit. So far, we discussed the model training and selection based on different parameterisations of the core model. To allow a comparison to other models, too, the entry point `baselines.py` builds a large set of established forecasting models, with a focus on the most relevant techniques applied in the statistics community [8]. These baselines range from naïve, seasonally corrected methods, over exponential smoothing, to seasonal auto-regressive methods.

2.2.7. Sample code snippet

The following sample code snippet and related visualisation, in Fig. 2, shows the transformation method of a `pandas` data frame to a `torch.tensor`. Through an allocation of the tensor to a graphical processing unit, PyTorch's semantics allow a significant acceleration of the training stage [25]. Nevertheless, *ProLoaF*'s implementation is device-agnostic and does not require a CUDA capable system.

```
config = utils.confighandler.read_config
(<config_path>)
train_data_loader, validation_data_loader,
test_data_loader =
  utils.datahandler.transform(<pandas.DataFrame>,
  device='gpu', **config)
```

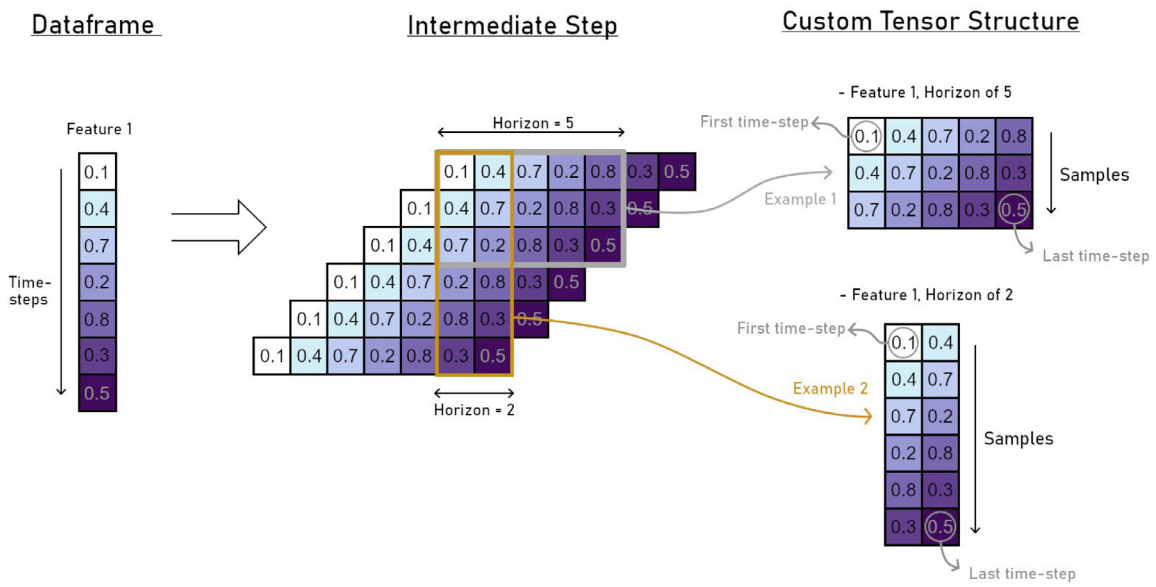

Fig. 2. Visualisation of the transform method: Data structure transformation from pandas data frame to PyTorch tensor as input for the forecast model.

3. Illustrative examples

The following short code snippet shows how a neural network can be loaded as pickled file from a given target directory and be used for predictions on new data.

```
net = torch.load(<model_path>)

targets, output =
  mh.get_prediction(net, test_data_loader,
  n_prediction_steps)

y_pred_upper, y_pred_lower, expected_values =
  mh.get_pred_interval(
  output, <scoring-method>, targets)
```

The forecast output is computed differently depending on the underlying scoring method, that was used during the training of the neural network model. In the case of the Gaussian Negative Log-Likelihood Score (GNLL), that is the default scoring method, we obtain two intermediate results per forecast time step to construct the prediction profiles. The first result is the mean value, while the second value is the standard deviation. The upper and lower bounds that are constructed by the `utils.modelhandler` method `get_pred_interval()` indicate the boundaries of the 95% confidence interval based on these two parameters.

The `utils.plot` module contains handy functions not only to visualise the forecasts at selected time steps, but also to plot forecast error distributions as shown in Fig. 3. The test data set consists of 356 days in hourly resolution.

```
with torch.no_grad():
  plot.plot_hist(
    targets,
    expected_values,
    y_pred_upper,
    y_pred_lower,
    analyzed_metrics=['residuals'],
    sample_frequency=1,
    save_to_disc=<path>)
```

The histogram of residuals in Fig. 3 also shows that the core forecast model outperforms highly-performant date-driven benchmark forecasting models, such as the earlier introduced *Prophet* [14]. The presented results are taken from a STLF case (24 h ahead) for the load in Germany. The underlying data from the open power system data (OPSD) platform, as well as the specific parameterisation and training process of each forecasting model is described in greater detail, in the project repository [26]. The notebook includes common baselines, such as persistence, naïve and ARIMA forecasts, among others.

4. Impact

The forecasting software package with the encoder–decoder method described in Section 2.2.3 has been deployed for continuous operational management in Northern-European distribution system operation control rooms. There, it has been used as a decision basis for energy management purposes on a day-ahead routine. Lessons learned from this first deployment are that the required degree of collaboration between the development, training, and deployment stages is very high. Many departments are collaboratively developing tools for the control room. To bring a forecast into operation, data collection and processing, feature engineering and the design of the model, its training and inference, and the actual utilisation are all steps which need to be closely coordinated. *ProLoaF* simplifies this process by covering the major software engineering tasks with the utils package and the data engineering tasks with the pre-defined high-level interfaces. The remaining adjustments are those that are specific for the forecast job which depends on the actual data from the target area. Thus, we move from a model centric development to a data centric development. In that sense, Proloaf offers the developer a holistic forecasting approach, which is one of the key requirements identified in Section 1.

Because the power system is a critical infrastructure, a human-in-the-loop step is to be considered for the evaluation and processing of the computed result in its daily operation. Given this application scenario, *ProLoaF* suggests an MLOps approach targeted for active power system operation and energy services for utilities. The specific intersection of ML, DevOps and power system engineering accelerates the scaling up of the adoption of ML models in daily business. Thereby, the focus shifts from further tuning newer and more complex models, which continuously emerge due to the fast developing ML domain, to more

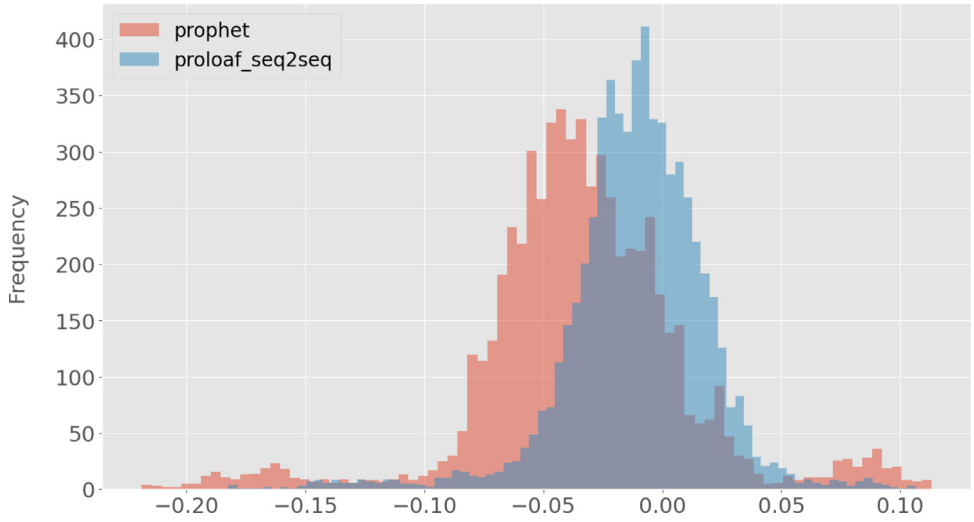

Fig. 3. Distribution of residuals.

data-centric work. With this streamlining and automation of the described loop, we expect to not only improve the performance of the forecasting models utilised in power system operation but also make the forecasts more scalable, reproducible and sustainable. *ProLoaF* achieves scalability through the formulation of the entry points. Reproducibility is ensured by the central config file and the logging functions. Finally, sustainability results from the many working hours saved, that would normally require specialists in each field. In the future, we aim to pursue further research questions on the basis of the human-in-the-loop aspect of *ProLoaF*. In particular, a thorough analysis on the value of forecast model quality and interpretability of the results are key questions to be further explored. To accelerate the deployment of this project and open the community for such research questions, which necessitate the know-how of stakeholders in the power system domain, *ProLoaF* has become one integral part of the Service-based Open-source Grid automation platform for Network Operation of the future (SOGNO) under the umbrella of Linux Foundation Energy.

5. Conclusions

ProLoaF is a Python based software project that links data and feature engineering, training, evaluation and benchmarking of time series forecasting models. To solve forecasting tasks relevant for power system services, it comprises typical entry points that are required in this context. All parametrisations for the data and on the forecasting model, are set through a central configuration file. In the future, the PyTorch based model library shall be further extended to complement the existing encoder–decoder class.

The forecast model, i.e., the forecasting core of the*ProLoaF* software has proven a high forecast quality compared to the most common baseline models during testing with exemplary data from the German transmission system [26] and in production of the Swedish sub-transmission system [22]. We emphasise that *ProLoaF* could in the future include various different forecasting models, as stated in Section 2.2.3. The developer team invites future developers and users to contribute with code and feedback to extend the functionalities and pursue research questions, e.g., on the user endpoint.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: The present work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957739 (OneNet project). The authors further declare that the financial support does not cause any conflicting interest with the work reported in this paper.

Data availability

The data is openly available via the given references in the open power system data platform as well as in a structured form in the published software repository.

Acknowledgements

The authors thank all contributors to the GitHub repository for their valuable contributions to this first release of the developed software, Ferdinanda Ponci for her continuous support, feedback and review, Miro Kolenic for his support in documentation, visualisation, and discussions, and Markus Mirz for the development of *ProLoaF* as a service in the LFE project SOGNO. Also, the authors gratefully acknowledge that the work leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957739 (OneNet project).

References

[1] Gürses-Tran Gonca. ProLoaF in SOGNO platform v0.1.1. 2021, http://dx.doi.org/10.5281/ZENODO.5171892, [Online accessed 31.01.2023].
[2] Beitner Jan. PyTorch Forecasting: Time series forecasting with PyTorch. 2020, URL https://pytorch-forecasting.readthedocs.io/en/stable/, [Online accessed 31.01.2023].
[3] Hong Tao, Fan Shu. Probabilistic electric load forecasting: A tutorial review. Int J Forecast 2016;32(3):914–38. http://dx.doi.org/10.1016/j.ijforecast.2015.11.011, [Online accessed 31.01.2023].
[4] Toubeau Jean-Francois, et al. Deep learning-based multivariate probabilistic forecasting for short-term scheduling in power markets. IEEE Trans Power Syst 2019;34(2):1203–15. http://dx.doi.org/10.1109/TPWRS.2018.2870041.
[5] Farrokhabadi Mostafa, et al. Day-ahead electricity demand forecasting competition: Post-COVID paradigm. IEEE Open Access J Power Energy 2022;9:185–91. http://dx.doi.org/10.1109/OAJPE.2022.3161101.
[6] Makridakis Spyros, Spiliotis Evangelos, Assimakopoulos Vassilios. M5 accuracy competition: Results, findings, and conclusions. Int J Forecast 2022;38(4):1346–64. http://dx.doi.org/10.1016/j.ijforecast.2021.11.013, Special Issue: M5 competition.
[7] Makridakis Spyros, Spiliotis Evangelos, Assimakopoulos Vassilios. The M4 competition: 100,000 time series and 61 forecasting methods. Int J Forecast 2020;36(1):54–74. http://dx.doi.org/10.1016/j.ijforecast.2019.04.014, M4 Competition.

[8] Hyndman RJ, Athanasopoulos G, editors. Forecasting: principles and practice. second print edition. Lexington, Ky: Otexts; 2018, [Online accessed 31.01.2023].
[9] Hyndman Rob J, Khandakar Yeasmin. Automatic time series forecasting: The forecast package for R. J Stat Softw 2008;27(3):1–22. http://dx.doi.org/10.18637/jss.v027.i03, [Online accessed 31.01.2023].
[10] Seabold Skipper, Perktold Josef. Statsmodels: Econometric and statistical modeling with python. In: Proceedings of the 9th python in science conference. Proceedings of the python in science conference, SciPy; 2010, p. 92–6. http://dx.doi.org/10.25080/Majora-92bf1922-011, [Online accessed 31.01.2023].
[11] Smith Taylor G, et al. pmdarima: ARIMA estimators for Python. 2017, URL http://www.alkaline-ml.com/pmdarima, [Online accessed 31.01.2023].
[12] Sheppard Kevin, et al. bashtage/arch: Release 5.0.1. 2021, http://dx.doi.org/10.5281/ZENODO.593254, [Online accessed 31.01.2023].
[13] Browell Jethro, Gilbert Ciaran. ProbCast: Open-source production, evaluation and visualisation of probabilistic forecasts. In: The 16th international conference on probabilistic methods applied to power systems, PMAPS 2020, 2020-08-18 - 2020-08-21, Liège, Belgium. IEEE; 2020, p. 1–6. http://dx.doi.org/10.1109/PMAPS47429.2020.9183441, [Online accessed 31.01.2023].
[14] Taylor Sean J, Letham Benjamin. Forecasting at scale. Amer Statist 2017;72:37–45. http://dx.doi.org/10.7287/peerj.preprints.3190v2, [Online accessed 31.01.2023].
[15] Alexandrov Alexander, et al. GluonTS: Probabilistic Time Series Models in Python. J Mach Learn Res 2022;21(1). http://dx.doi.org/10.48550/arXiv.1906.05264, [Online accessed 31.01.2023].
[16] Triebe Oskar, Laptev Nikolay, Rajagopal Ram. AR-Net: A simple Auto-Regressive Neural Network for time-series. 2019, http://dx.doi.org/10.48550/arXiv.1911.12436, [Online accessed 31.01.2023].
[17] Muehlenpfordt Jonathan. Open power system data. 2020. Data package time series. Version 2020-10-06. 2020, http://dx.doi.org/10.25832/TIME_SERIES/2020-10-06, [Online accessed 31.01.2023].
[18] MLOps Special Interest Group. MLOps Roadmap 2021. 2021, URL https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2021/MLOpsRoadmap2021.md, [Online accessed 31.01.2023].
[19] Pedregosa Fabian, et al. Scikit-learn: Machine Learning in Python. J Mach Learn Res 2011;12(85):2825–30, URL http://jmlr.org/papers/v12/pedregosa11a.html, [Online accessed 31.01.2023].
[20] Barandas Marília, et al. TSFEL: Time series feature extraction library. SoftwareX 2020;11:100456. http://dx.doi.org/10.1016/j.softx.2020.100456, [Online accessed 31.01.2023].
[21] Gürses-Tran Gonca, Flamme Hendrik, Monti Antonello. Probabilistic load forecasting for day-ahead congestion mitigation. In: The 16th international conference on probabilistic methods applied to power systems, PMAPS 2020, 2020-08-18 - 2020-08-21, Liège, Belgium. 2020, p. 1–6. http://dx.doi.org/10.1109/PMAPS47429.2020.9183670, [Online accessed 31.01.2023].
[22] Bjarup David, Isendahl Christopher. Finalized small scale demo run – functionalities, products and routines. 2021, http://dx.doi.org/10.3030/824414, [Online accessed 31.01.2023].
[23] Akiba Takuya, et al. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining. 2019, http://dx.doi.org/10.1145/3292500.3330701, [Online accessed 31.01.2023].
[24] TensorFlow Developers. TensorFlow: Large-scale machine learning on heterogeneous systems. 2021, http://dx.doi.org/10.5281/ZENODO.4724125, [Online accessed 31.01.2023].
[25] Paszke Adam, et al. PyTorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems 32. Curran Associates, Inc; 2019, p. 8024–35. http://dx.doi.org/10.5555/3454287.3455008, [Online accessed 31.01.2023].
[26] Gürses-Tran Gonca. Notebook - train and benchmark models. 2021, URL https://github.com/sogno-platform/proloaf/blob/master/notebooks/Train%20and%20Benchmark%20Models.ipynb, [Online accessed 31.01.2023].