

***BlackBox Toolkit: Intelligent Assistance to UI Design***

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der  
RWTH Aachen University zur Erlangung des akademischen Grades  
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

*Vinoth Pandian Sermuga Pandian, M.Sc.*

aus

*Sivakasi, India*

Berichter: *Prof. Dr. Matthias Jarke*  
*Prof. Dr. Ulrich J. Schröder*

Tag der mündlichen Prüfung: 15.02.2022

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.





## Eidesstattliche Erklärung

I, *Vinoth Pandian Sermuga Pandian*

erklärt hiermit, dass diese Dissertation und die darin dargelegten Inhalte die eigenen sind und selbstständig, als Ergebnis der eigenen originären Forschung, generiert wurden.

Hiermit erkläre ich an Eides statt

1. Diese Arbeit wurde vollständig oder größtenteils in der Phase als Doktorand dieser Fakultät und Universität angefertigt;
2. Sofern irgendein Bestandteil dieser Dissertation zuvor für einen akademischen Abschluss oder eine andere Qualifikation an dieser oder einer anderen Institution verwendet wurde, wurde dies klar angezeigt;
3. Wenn immer andere eigene- oder Veröffentlichungen Dritter herangezogen wurden, wurden diese klar benannt;
4. Wenn aus anderen eigenen- oder Veröffentlichungen Dritter zitiert wurde, wurde stets die Quelle hierfür angegeben. Diese Dissertation ist vollständig meine eigene Arbeit, mit der Ausnahme solcher Zitate;
5. Alle wesentlichen Quellen von Unterstützung wurden benannt;
6. Wenn immer ein Teil dieser Dissertation auf der Zusammenarbeit mit anderen basiert, wurde von mir klar gekennzeichnet, was von anderen und was von mir selbst erarbeitet wurde;
7. Ein Teil oder Teile dieser Arbeit wurden zuvor veröffentlicht und zwar in:

*Suleri, S., **Pandian, V.P.S.**, Shishkovets, S., Jarke, M., 2019. Eve: A Sketch-based Software Prototyping Workbench, in: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI EA '19. ACM, New York, NY, USA, p. Lbw1410:1-lbw1410:6. <https://doi.org/10.1145/3290607.3312994>*

**Pandian, V.P.S.**, Suleri, S., 2020. BlackBox toolkit: Intelligent assistance to UI design. *arXiv e-prints arXiv-2004.*

**Pandian, V.P.S., Suleri, S., Jarke, M., 2020.** Syn: Synthetic Dataset for Training UI Element Detector From Lo-Fi Sketches, in: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion, IUI '20*. Association for Computing Machinery, Cagliari, Italy, pp. 79–80. <https://doi.org/10.1145/3379336.3381498>

**Pandian, V.P.S., Suleri, S., Beecks, C., Jarke, M., 2020.** MetaMorph: AI Assistance to Transform Lo-Fi Sketches to Higher Fidelities, in: *32nd Australian Conference on Human-Computer Interaction, OzCHI '20*. Association for Computing Machinery, New York, NY, USA, pp. 403–412. <https://doi.org/10.1145/3441000.3441030>

**Pandian, V.P.S., Suleri, S., Jarke, M., 2021.** UISketch: A large-scale dataset of UI element sketches, in: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI '21*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3411764.3445784>

**Pandian, V.P.S., Suleri, S., Jarke, M., 2021.** SynZ: Enhanced synthetic dataset for training UI element detectors, in: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion, IUI '21*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3397482.3450725>

**Rahman, S., Pandian, V.P.S., Jarke, M., 2021.** RUIE: Refining UI layout aesthetics using transformer encoder, in: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion, IUI '21*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3397482.3450716>

**Gajjar, N., Pandian, V.P.S., Suleri, S., Jarke, M., 2021.** Akin: Generating UI wireframes from UI design patterns using deep learning, in: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion, IUI '21*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3397482.3450727>

**Pandian, V.P.S., Shams, A., Suleri, S., Jarke, M., 2021.** LoFi Sketch: A Large Scale Dataset of Smartphone Low Fidelity Sketches, in: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22*. ACM, New York, NY, USA. <https://doi.org/10.1145/3491101.3519624>

Datum: 15.02.2022



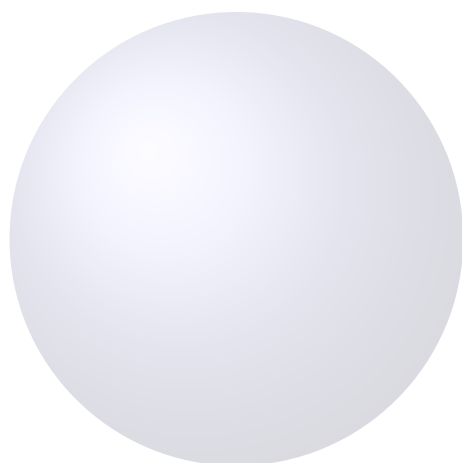
Vinoth Pandian Sermuga Pandian

---

## DEDICATION

---

*To my parents* who always believed I will be able to finish this work, and  
*to my wife* who made me finish this work.





---

## ABSTRACT

---

User Interface (UI) design is an iterative process where designers iterate over multiple prototyping fidelities to finalise an aesthetic and usable interface. Prior research on adding Artificial Intelligence (AI) to the UI design process focused on automating the process while sacrificing the autonomy of designers. In this dissertation, we conduct systematic research using a human-centred approach to provide AI assistance to UI designers before, during, and after the traditional LoFi prototyping process. As a result, this research aims to provide coherent AI assistance throughout the repetitive and arduous LoFi prototyping task without sacrificing the autonomy of UI designers. In doing so, we contribute the BlackBox Toolkit. This toolkit assists designers by creating four large-scale, diverse, open-access benchmark datasets and three AI tools that assist UI designers throughout the LoFi prototyping process.

Blackbox toolkit contributes the following datasets: UISketch dataset, ~18k UI element sketches; Syn & SynZ datasets, ~300k synthetic LoFi sketches; LoFi Sketch dataset, ~4.5k real-life LoFi sketches and Wired dataset, ~2.7k semantically annotated UI screenshots. Each of these datasets targets one of the two types of LoFi prototypes: LoFi sketches and LoFi wireframes. The datasets ensure ample diversity of designers and developers by data collection from a wide range of countries, input media, and prior experience.

Moving on to the AI tools, Akin is a UI wireframe generator that uses a modified conditional SAGAN model and assists UI designers before LoFi prototyping by generating multiple UI wireframes for a given UI design pattern. Evaluation results show that the quality of Akin-generated UI wireframes was adequate. Further, the user evaluation shows that UI/UX designers considered Akin-generated wireframes as good as designer-made wireframes. Further, designers identified Akin-generated wireframes as designer-made 50% of the time.

RUIITE is a UI wireframe refiner that uses a Transformer-Encoder model and assists UI designers during LoFi prototyping by aligning and grouping UI elements in a given UI wireframe. On almost all evaluation metrics, it provides satisfactory results. The qualitative feedback indicates that designers prefer UI wireframe refinement using RUIITE and expressed interest in using it.

MetaMorph is a UI element detector that uses the RetinaNet object detection model and assists UI designers after LoFi prototyping by detecting the constituent UI elements of LoFi sketches and their location and dimension. It enables the transformation of LoFi sketches to higher-fidelities. Upon evaluation with hand-drawn LoFi sketches, MetaMorph provides 47.8% mAP. Further, the qualitative feedback shows that MetaMorph reduced designers' effort in transforming LoFi prototype to higher fidelities by providing them with a head-start.

In summary, from our qualitative feedback, the UI designers perceive utilising AI for UI design as an exciting and practical approach and expressed their eagerness to adopt such tools. Moreover, the user satisfaction studies conducted using After Scenario Questionnaires show an above-average designer satisfaction level upon using all three AI assistance tools.

This research aims to understand the impact of AI tools in UI designer workflow and assess their satisfaction upon using these AI tools. Further, it sets a baseline for future research on UI wireframe generation, refinement and transformation.



---

## ZUSAMMENFASSUNG

---

Das Design von Benutzeroberflächen (UI) ist ein iterativer Prozess, bei dem die Designer mehrere Prototyping-Fidelity durchlaufen, um eine ästhetische und benutzbare Oberfläche zu erstellen. Frühere Forschungen zur Integration von Künstlicher Intelligenz (KI) in den UI-Designprozess konzentrierten sich auf die Automatisierung des Prozesses, wobei die Autonomie der Designer auf der Strecke blieb. In dieser Dissertation führen wir eine systematische Forschung durch, die einen menschenzentrierten Ansatz verwendet, um KI-Unterstützung für UI-Designer vor, während und nach dem traditionellen LoFi-Prototyping-Prozess bereitzustellen. Diese Forschung zielt darauf ab, kohärente KI-Unterstützung während der sich wiederholenden und mühsamen LoFi-Prototyping-Aufgabe zu bieten, ohne die Autonomie der UI-Designer zu opfern. Zu diesem Zweck stellen wir das Blackbox Toolkit zur Verfügung. Dieses Toolkit unterstützt Designer durch die Erstellung von vier großen, vielfältigen, frei zugänglichen Benchmark-Datensätzen und drei KI-Tools, die UI-Designer während des gesamten LoFi-Prototyping-Prozess unterstützen.

Das Blackbox-Toolkit steuert die folgenden Datensätze bei: UISketch-Datensatz, ~18K Skizzen von UI-Elementen; Syn & SynZ-Datensätze, ~300K synthetische LoFi-Skizzen; LoFi Sketch Datensatz, ~4.5K reale LoFi-Skizzen und Wired-Datensatz, ~2.7K semantisch annotiert UI-Screenshots. Jeder dieser Datensätze zielt auf eine der beiden Arten von LoFi-Prototypen ab: LoFi-Skizzen und LoFi-Wireframes. Die Datensätze gewährleisten eine große Vielfalt an Designern und Entwicklern durch die Datenerfassung aus einer Vielzahl von Ländern, Eingabemedien und Vorerfahrungen.

Akin ist ein UI-Wireframe-Generator, der ein modifiziertes bedingtes SAGAN-Modell verwendet und UI-Designer vor dem LoFi-Prototyping unterstützt, indem er mehrere UI-Wireframes für ein bestimmtes UI-Designmuster erzeugt. Die Evaluierungsergebnisse zeigen, dass die Qualität der von Akin generierten UI-Wireframes angemessen war. Außerdem zeigt die Nutzerbewertung, dass UI/UX-Designer die von Akin generierten Wireframes für genauso gut halten wie die von Designern erstellten Wireframes. Außerdem identifizierten die Designer die von Akin generierten Wireframes in 50% der Fälle als vom Designer erstellt.

RUIITE ist ein UI-Wireframe-Refiner, der ein Transformer-Encoder-Modell verwendet und UI-Designer beim LoFi-Prototyping unterstützt, indem er UI-Elemente in einem bestimmten UI-Wireframe ausrichtet und gruppiert. Bei fast allen Bewertungsmetriken liefert es zufriedenstellende Ergebnisse. Das qualitative Feedback deutet darauf hin, dass die Designer die Verfeinerung des UI-Wireframes mit RUIITE bevorzugen und ihr Interesse an dessen Einsatz bekundet haben.

MetaMorph ist ein UI-Element-Detektor, der das RetinaNet-Objektdetektion Modell verwendet und UI-Designer nach dem LoFi-Prototyping unterstützt, indem er die konstituierenden UI-Elemente von LoFi-Skizzen sowie deren Position und Dimension erkennt. Es ermöglicht die Transformation von LoFi-Skizzen in eine höhere Fidelity Prototypen. Bei der Evaluierung mit handgezeichneten LoFi-Skizzen erreicht MetaMorph 47,8% mAP. Darüber hinaus zeigt das qualitative Feedback, dass MetaMorph den Aufwand der Designer bei der Umwandlung von LoFi-Prototypen in höhere Fidelitäten reduziert, indem es ihnen einen Vorsprung verschafft.

Zusammenfassend lässt sich aus unserem qualitativen Feedback schließen, dass die Benutzeroberflächendesigner den Einsatz von KI für das Benutzeroberflächendesign als einen spannenden und praktischen Ansatz ansehen und sich sehr für den Einsatz solcher Tools interessieren. Darüber hinaus zeigen die Studien zur Benutzerzufriedenheit, die mithilfe von After Scenario Fragebögen durchgeführt wurden, eine überdurchschnittlich hohe Zufriedenheit der Designer mit allen drei KI-Tools.

Ziel dieser Forschung ist es, die Auswirkungen von KI-Tools auf den Arbeitsablauf von UI-Designern zu verstehen und ihre Zufriedenheit mit dem Einsatz dieser KI-Tools zu bewerten. Darüber hinaus wird eine Grundlage für künftige Forschungen zur Erstellung, Verfeinerung und Transformation von UI-Wireframes geschaffen.





---

## CONTENTS

---

1	INTRODUCTION	1
1.1	Low Fidelity Prototyping . . . . .	2
1.2	Automating UI prototyping . . . . .	4
1.3	Human Centered AI . . . . .	5
1.4	Thesis Statement . . . . .	5
1.5	Scope . . . . .	5
1.6	Research Questions . . . . .	6
1.7	Research Design . . . . .	6
1.8	Research Contributions . . . . .	8
1.9	Outline . . . . .	10
2	LITERATURE REVIEW	13
2.1	AI support in UI prototyping . . . . .	13
2.2	UI Datasets . . . . .	17
2.3	Automated Transformation of Prototype Fidelity . . . . .	19
2.4	UI Wireframe Generation and Refinement . . . . .	20
2.5	GUI similarity search . . . . .	22
2.6	Identified Research Gaps . . . . .	23
2.7	Summary . . . . .	23
3	BLACKBOX TOOLKIT	25
3.1	Goals . . . . .	25
3.2	Blackbox Toolkit . . . . .	26
3.3	Datasets . . . . .	26
3.4	AI tools . . . . .	27
<b>I</b>	<b>DATASETS</b>	31
4	UISKETCH DATASET	33
4.1	Objective . . . . .	33
4.2	Taxonomy of UI element sketches . . . . .	34
4.3	Data collection questionnaires . . . . .	34
4.4	Pilot study & Design decisions . . . . .	35
4.5	Participants . . . . .	36
4.6	Procedure . . . . .	37
4.7	Data processing . . . . .	38
4.8	Data verification . . . . .	39

4.9	Collected data . . . . .	40
4.10	Applications . . . . .	41
4.11	Summary . . . . .	41
5	LOFI SKETCH DATASET	43
5.1	Objective . . . . .	43
5.2	Design decisions . . . . .	44
5.3	Data collection questionnaires . . . . .	44
5.4	Participants . . . . .	45
5.5	Procedure . . . . .	46
5.6	Data verification . . . . .	48
5.7	Data annotation . . . . .	49
5.8	Collected data . . . . .	49
5.9	Benefits & Applications . . . . .	50
5.10	Summary . . . . .	51
6	SYN & SYNZ DATASETS	53
6.1	Objective . . . . .	53
6.2	Syn . . . . .	53
6.3	SynZ . . . . .	56
6.4	Comparison of Syn and SynZ datasets . . . . .	59
6.5	Benefits & Applications . . . . .	60
6.6	Summary . . . . .	61
7	WIRED DATASET	63
7.1	Objective & Design Decisions . . . . .	64
7.2	Data classification . . . . .	65
7.3	Data annotation . . . . .	67
7.4	Collected data . . . . .	68
7.5	Benefits & Applications . . . . .	68
7.6	Summary . . . . .	69
<b>II</b>	<b>FORMATIVE ANALYSIS</b>	<b>71</b>
8	HUMAN RECOGNITION STUDY	73
8.1	Participants . . . . .	73
8.2	Measurements . . . . .	73
8.3	Apparatus . . . . .	74
8.4	Procedure . . . . .	74
8.5	Analysis . . . . .	75
8.6	Results & Discussion . . . . .	75
8.7	Summary . . . . .	79
9	COMPUTER RECOGNITION STUDY	81

9.1	DNN models . . . . .	81
9.2	Preprocessing . . . . .	82
9.3	Training . . . . .	82
9.4	Evaluation . . . . .	83
9.5	Results & Discussion . . . . .	84
9.6	Human vs Computer recognition . . . . .	87
9.7	Analysis of UI element sketches . . . . .	88
9.8	Summary . . . . .	89
<b>III</b>	<b>AKIN</b>	91
10	BACKGROUND & PROPOSED SOLUTION	93
10.1	Background . . . . .	94
10.2	Identified Gaps . . . . .	94
10.3	Proposed Solution . . . . .	95
11	MODEL ARCHITECTURE & DATA REPRESENTATIONS	97
11.1	Model Architectures . . . . .	97
11.2	Data Representations . . . . .	99
12	IMPLEMENTATION	103
12.1	Implementation Approach . . . . .	103
12.2	Datasets . . . . .	104
12.3	Implementation Details . . . . .	104
12.4	Configuration & Training Process . . . . .	107
12.5	Post processing steps . . . . .	107
13	EVALUATION	109
13.1	Metrics . . . . .	109
13.2	Methodology . . . . .	110
14	USER EVALUATION	113
14.1	User Evaluation of UI wireframes . . . . .	113
14.2	Study Design & Measurements . . . . .	114
14.3	User satisfaction study . . . . .	120
14.4	Methodology . . . . .	121
15	SUMMARY & FUTURE WORK	127
<b>IV</b>	<b>RUI TE</b>	129
16	PROPOSED SOLUTION	131
16.1	Background . . . . .	131
16.2	Identified Gaps . . . . .	132
16.3	Proposed Solution . . . . .	133
17	MODEL ARCHITECTURES & DATA REPRESENTATIONS	135

17.1	Model architectures . . . . .	135
17.2	Data Representations . . . . .	137
17.3	Discrete Sequential Representation . . . . .	139
18	IMPLEMENTATION . . . . .	143
18.1	Implementation Approach . . . . .	143
18.2	Datasets . . . . .	143
18.3	Implementation Details . . . . .	144
18.4	Configuration & Training Process . . . . .	145
18.5	Post processing steps . . . . .	145
19	EVALUATION . . . . .	147
19.1	Dataset . . . . .	147
19.2	Metrics . . . . .	147
19.3	Methodology . . . . .	149
19.4	Results . . . . .	149
20	USER EVALUATION . . . . .	151
20.1	Web API & Adobe XD plugin . . . . .	151
20.2	Methodology . . . . .	152
20.3	Participants . . . . .	152
21	SUMMARY & FUTURE WORK . . . . .	157
<b>V</b>	<b>METAMORPH . . . . .</b>	<b>159</b>
22	BACKGROUND & PROPOSED SOLUTION . . . . .	161
22.1	Background . . . . .	161
22.2	Identified Gaps . . . . .	162
22.3	Proposed Solution . . . . .	162
23	MODEL ARCHITECTURES & DATA REPRESENTATIONS . . . . .	165
23.1	Model Architectures . . . . .	165
23.2	Data Representations . . . . .	166
24	IMPLEMENTATION . . . . .	169
24.1	Implementation Approach . . . . .	169
24.2	Datasets . . . . .	169
24.3	Implementation Details . . . . .	170
24.4	Configuration . . . . .	170
24.5	Training Process . . . . .	171
25	EVALUATION . . . . .	175
25.1	Dataset . . . . .	175
25.2	Metrics . . . . .	175
25.3	Methodology . . . . .	176
25.4	Results . . . . .	177

25.5 Discussion . . . . .	179
26 USER EVALUATION	181
26.1 Web API . . . . .	181
26.2 Eve: Prototyping Workbench . . . . .	181
26.3 Methodology . . . . .	182
26.4 Results & Discussion . . . . .	184
27 SUMMARY & FUTURE WORK	187
28 CONCLUSION & FUTURE WORK	189
<b>VI APPENDIX</b>	193
A COMPREHENSIVE LIST OF LITERATURE	195
B SEMANTIC ANNOTATIONS OF UI SCREENSHOTS	201
 BIBLIOGRAPHY	 205
 PUBLICATIONS	 221



---

## LIST OF FIGURES

---

Figure 1.1	Sample LoFi prototypes of a UI screen . . . . .	2
Figure 1.2	Overview of the datasets and AI tools in the Black-Box toolkit . . . . .	7
Figure 3.1	Akin assists UI designers before they start prototyping by generating UI wireframes for the chosen UI design pattern. It is trained and evaluated with the Wired dataset . . . . .	27
Figure 3.2	RUI TE assists UI designers during prototyping by aligning and grouping the UI wireframe layout. It is trained and evaluated with the SynZ dataset annotations . . . . .	28
Figure 3.3	MetaMorph assists UI designers after the LoFi prototyping by detecting the constituent UI elements thereby enabling transformation to higher fidelities. This tool is pretrained with the synthetic LoFi sketches from the Syn and SynZ datasets. It is further fine-tuned and evaluated with the LoFi Sketch dataset . . . . .	29
Figure 4.1	Paper and digital questionnaire used for collecting UISketch dataset. Both questionnaires have an example of a UI element on the left and a rectangular area to sketch the UI element on the right-hand side. . . . .	36
Figure 4.2	Demographics of participants as percentage of participants per country . . . . .	37
Figure 4.3	Count of UI element sketches collected from different type of participants, split by the medium that they used to sketch . . . . .	38
Figure 4.4	Process of extracting the close-cropped UI element sketches from the paper questionnaire . . .	39
Figure 4.5	Examples of rejected sketches due to contamination, badly drawn or irrelevant content. . . . .	39

Figure 4.6	Distribution of digitally drawn UI element sketches using stylus vs. paper sketches drawn using pen and pencil in UISketch dataset . . . . .	40
Figure 5.1	Paper and digital questionnaire used for collecting LoFi sketch dataset. Both questionnaires have an example of a inspiration UI screenshot on the left and a rectangular area to sketch the LoFi on the right-hand side. . . . .	45
Figure 5.2	Count of participants ( $\geq 5$ people) per country that contributed to LoFi sketch dataset . . . . .	46
Figure 5.3	Samples of LoFi sketches rated in a scale of 1 to 5. . . . .	48
Figure 5.4	Sample LoFi Sketch with its annotations . . . . .	49
Figure 5.5	Distribution of digitally drawn LoFi sketches using stylus, mouse, and touch vs. paper sketches drawn using pen and pencil in LoFi sketch dataset . . . . .	50
Figure 6.1	Sample generated synthetic LoFi sketches from the Syn dataset . . . . .	54
Figure 6.2	Flowchart visualizing different steps involved in modifying RICO annotations to SynZ dataset . . . . .	56
Figure 6.3	Sample generated synthetic LoFi sketches from the SynZ dataset . . . . .	58
Figure 6.4	Sample representative UI images from (a) Syn, (b) SynZ, (c) RICO . . . . .	59
Figure 6.5	Comparision of UI element distribution using in Syn vs SynZ represented by a histogram plot . . . . .	60
Figure 7.1	A sample image of each UI design pattern from the dataset . . . . .	65
Figure 7.2	Sample annotations with most common categories of semantic UI elements . . . . .	68
Figure 8.1	Screenshot of the web application to collect data for human recognition. We show a random UI element sketch from our UISketch dataset on the left and a list of 21 UI element categories on the right-hand side . . . . .	74
Figure 8.2	Confusion matrix for UI element categories on human recognized category vs the original category . . . . .	77
Figure 8.3	Representative sketches from 21 UI element categories with highest (left) and lowest (right) human recognition accuracy . . . . .	77



Figure 8.4	UI element categories confused by UI/UX designers due to their semantic similarities . . . . .	78
Figure 8.5	UI element categories confused by UI/UX designers due to their structural similarities . . . . .	78
Figure 9.1	Confusion matrix for UI element categories on ResNet 152 predictions vs the ground truth . . .	85
Figure 9.2	UI element categories with structural similarities	87
Figure 9.3	t-SNE visualization of representative sketches of 21 categories of UI elements from the test dataset of UISketch dataset. We have manually marked 13 clusters based on the proximity of these clusters.	88
Figure 9.4	Representative images showing the activated pixel areas and the underlying structure of all UI element categories. . . . .	89
Figure 10.1	Akin is UI wireframe generator that generates UI wireframes for the chosen UI design pattern . .	93
Figure 11.1	Sample UI screenshot for each of the 5 UI design pattern in the Wired dataset along with their semantic images . . . . .	99
Figure 11.2	Samples of semantic UI wireframes generated by trained SAGAN and PixelCNN++ model for each of the 5 UI design pattern . . . . .	100
Figure 11.3	Sample layout vector representation of a login UI design pattern . . . . .	101
Figure 11.4	Samples of layout vector UI wireframes generated by trained Transformer-Encoder model for each of the 5 UI design pattern . . . . .	102
Figure 12.1	Architecture of Akin's conditional SAGAN model	105
Figure 12.2	Akin is a UI wireframe generator trained with the semantic representations of UI images of 5 different UI design patterns from the Wired dataset. Akin's trained generator model (in the pale red box) generates different UI wireframes for a chosen UI design pattern. . . . .	106
Figure 12.3	Conversion process of sample Akin generated semantic images to UI wireframes . . . . .	108

Figure 14.1	Screenshots of the web application to conduct rapid scene categorization and rating-preference judgment studies before and after the participant clicks "Show Wireframe for 500 ms" button . . .	115
Figure 14.2	Confusion matrix for rapid scene categorization study comparing participants' accuracy of identifying a given UI wireframe as Akin-generated or designer-made . . . . .	117
Figure 14.3	Percentage of Akin-generated UI wireframes classified by the participants as designer-made for the given UI wireframe presentation time intervals	118
Figure 14.4	Distribution of participants' subject rating of Akin-generated and designer-made UI wireframes for all UI design pattern types . . . . .	119
Figure 14.5	Distribution of participants who considered whether Akin-generated UI wireframes of given UI design pattern type contains all essential UI types . . .	120
Figure 14.6	Screenshot of Akin plugin and the generated wireframes in Adobe XD prototyping tool . . . . .	121
Figure 14.7	Spearman rank correlation between UI element detection precision and the three questions from ASQ . . . . .	124
Figure 16.1	RUITE improves the UI layout alignment by optimizing the location and dimension of UI elements on a screen. It takes a list of UI elements and their respective bounding boxes as input and optimizes the layout to be aesthetically appealing using a Transformer Encoder . . . . .	131
Figure 17.1	Rule-base equations for converting different positions of UI elements <b>A</b> and <b>B</b> from a UI wireframe to an adjacency matrix for UI wireframe graph representation . . . . .	138
Figure 17.2	Sample UI wireframe and its corresponding graph representation . . . . .	139
Figure 17.3	Process of discretizing a UI element bounding box by assigning it to grid and then converting it to a sequence . . . . .	139
Figure 17.4	Process of converting a UI wireframe to a sequence	140
Figure 18.1	Architecture of RUITE's Transformer-Encoder model	145

Figure 19.1	mAP score of RUIITE model during evaluation for different IoU thresholds ranging from 5% to 95%	150
Figure 20.1	Screenshot of RUIITE plugin and the refined wire-frame in Adobe XD prototyping tool . . . . .	151
Figure 20.2	Spearman rank correlation between UI element detection precision and the three questions from ASQ . . . . .	155
Figure 22.1	MetaMorph takes a LoFi sketch ( <b>left</b> ) as input and detects UI elements ( <b>right</b> ) using DNNs. For each detected UI element, it provides the object's category, prediction probability, and bounding box details (x,y coordinates of the top left corner, width, and height). . . . .	161
Figure 23.1	LoFi sketches represented as labelled UI element sketches for training the classification DNN . . .	167
Figure 23.2	LoFi sketches represented as an image and annotation file with the list of constituent UI element categories, their location and dimension . . . .	168
Figure 24.1	Architecture of the SSD Resnet 50 (Retinanet) model from the Detectron2 library as depicted by Honda (2020) . . . . .	170
Figure 24.2	Training loss per steps of the RetinaNet model during training with the LoFi sketches from the Syn and SynZ training datasets . . . . .	172
Figure 24.3	Training loss per steps of the RetinaNet model during training with the 3,632 LoFi sketches from the LoFi sketch training dataset . . . . .	172
Figure 25.1	COCO detection metrics of MetaMorph UI element detector measured with the hand-drawn LoFi sketch evaluation dataset . . . . .	177
Figure 25.2	Examples of UI element detection by MetaMorph using hand-drawn LoFi sketches of the user profile, images gallery, scale sheet, and settings screen.	178
Figure 25.3	UI element category wise average precision of MetaMorph UI element detector . . . . .	178
Figure 26.1	Eve: LoFi to MeFi transformation using UI element detection provided by MetaMorph (Pandian, Suleri, Beecks, et al., 2020) . . . . .	182

Figure 26.2	Spearman rank correlation between UI element detection precision and the three questions from ASQ . . . . .	185
-------------	---	-----

---

## LIST OF TABLES

---

Table 2.1	Split of literature based on the prototype fidelity, research focus and AI method applied . . . . .	14
Table 2.2	Comprehensive list of research on AI support for UI prototyping . . . . .	15
Table 2.3	List of UI datasets and their attributes . . . . .	17
Table 4.1	Overview of data collection and verification process in terms of quantity of UI element sketches collected and extracted . . . . .	40
Table 4.2	Count of sketches collected for each UI element category . . . . .	40
Table 5.1	Split of participants and count of collected sketches based on data collection source . . . . .	47
Table 5.2	Overview of sketches removed during LoFi sketch data verification process . . . . .	48
Table 6.1	Comparison of Syn and SynZ datasets . . . . .	60
Table 7.1	Count of UI screenshots per UI design pattern and UI element annotations per UI screenshot in the dataset . . . . .	66
Table 7.2	Mean, standard deviation and mode of the number of UI elements per UI screenshot in each UI design pattern category . . . . .	67
Table 8.1	Classification report of human detection accuracy, precision, recall, and F1-score . . . . .	76
Table 8.2	Human detection accuracy for semantically similar UI elements . . . . .	78
Table 8.3	Human detection accuracy for structurally similar UI elements . . . . .	78

Table 9.1	UISketch test dataset prediction accuracy and top 5 accuracy of each computational classification models along with their published year and total number of parameters . . . . .	84
Table 9.2	Classification report of ResNet 152 model's accuracy, precision, recall, and F1-score . . . . .	86
Table 9.3	Computational detection accuracy for structurally similar UI elements . . . . .	87
Table 14.1	Results of rapid scene categorization study across all five intervals . . . . .	118
Table 14.2	Results of rating-preference judgement study for all UI design pattern types . . . . .	119
Table 14.3	Percentage of participants who considered whether Akin-generated UI wireframes of given UI design pattern type contains all essential UI types . . .	120
Table 14.4	Results of the ASQ study with participant's prior experience in UI prototyping . . . . .	123
Table 19.1	mAP scores of RUIITE for different IoU threshold values . . . . .	150
Table 20.1	Results of the ASQ study with participant's prior experience in UI prototyping . . . . .	154
Table 25.1	Distribution of UI element category in the 895 LoFi sketches in the LoFi sketch evaluation dataset	176
Table 25.2	COCO detection metrics of MetaMorph object detection model measured with the hand-drawn LoFi sketch evaluation dataset . . . . .	177
Table 25.3	COCO detection metrics of MetaMorph object detection model measured with the hand-drawn LoFi sketch evaluation dataset . . . . .	179
Table 26.1	The number of UI elements the participants sketched, the number of elements MetaMorph correctedly identified, wrongly identified and the number of unidentified UI elements during qualitative study using Eve along with their ASQ results . . . . .	184
Table a.1	List of literature that provide AI support in UI design by automating the prototype fidelity transformation . . . . .	196

Table a.2	List of literature that provide AI support in UI design by generating and refining LoFi UI wireframes or MeFi prototypes . . . . .	199
Table a.3	List of literature that provide AI support in UI design by searching similar UI screenshots of given LoFi sketch or MeFi prototype . . . . .	200
Table a.4	List of literature that provide AI support in UI design by autocompleting MeFi prototype during MeFi design process . . . . .	200
Table b.1	Exhaustive list of semantic UI element categories and its equivalent UISketch UI element category	201

---

## INTRODUCTION

---

User Interface (UI) is an essential component of a software application. It acts as a bridge between two different realms: the creative and irrational humans; and the unimaginative and logical world of machines. UI designers act as the architect of this bridge by designing interfaces for humans while fitting their designs into the confinements and restrictions placed by machines' limitations. However, in the end, a UI is created for the users, so the core of UI design, as Dix (2004) describes, is to "put the user first, keep the user in the center and remember the user at the end." However, designing a UI appropriate for users while confining to machine limitations is an arduous task. Therefore, the universally followed and recommended approach in UI design is to create prototypes and iteratively improve them by evaluating them with users to attain the best possible design within the given time and budget limits (Dix, 2004; Shneiderman and Plaisant, 2004; Wilson et al., 1988). These prototypes serve as the foundation for building the end product—usable software.

During the UI prototyping process, designers go through multiple fidelities of prototypes. Starting from low-fidelity (LoFi) freehand sketches or wireframes to medium-fidelity (MeFi) digital images and finally to high-fidelity (HiFi) interactive UI screens or code (Engelberg et al., 2002). The different prototyping fidelities have their respective strengths and weaknesses.

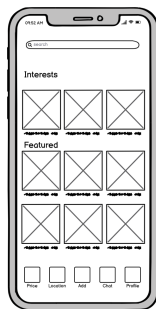
A LoFi prototype is cheap and supports ideating different designs quickly; however, it does not faithfully depict the system's final look-and-feel (Rudd et al., 1996). Similarly, MeFi contains the most necessary information, such as layouts and colours, and is relatively quick to create than HiFi; however, it is expensive and time-consuming to create multiple design variations compared to LoFi (Engelberg et al., 2002). HiFi resembles the final product, but the workload of creating such a system is humongous, and it is laborious to create multiple designs or modifications to the prototype

(Rudd et al., 1996). In addition to prototyping, UI designers also have to evaluate the prototypes with users to understand whether their designs are acceptable and usable (Dix, 2004). From the issues they uncover during this evaluation, they reiterate and modify their prototypes until they create a user-friendly UI design.

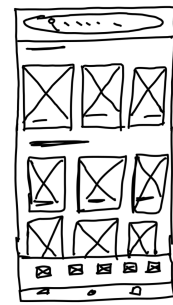
These widely used UI prototyping practices heavily relies on designers creating multiple UI designs for a software application. Dix (2004) explains finding the best possible UI design for a given task as a hill-climbing problem where the designers have to find the peak of a hill that represents the best possible design solution for a given system. Even if designers start at any point in terrain and iterate repeatedly, they might get stuck at a hill's local maxima instead of its global peak. Therefore, it is generally recommended to diverge in the initial stages (like parachuting in multiple places on terrain) and then converge (while abandoning a few) to finally reach the best possible design. The most eligible candidate for such an approach is LoFi prototyping, and therefore, as (Dix, 2004) asserts, it is the initial, fundamental, and indispensable step in creating UI designs.

### 1.1 LOW FIDELITY PROTOTYPING

The primary purpose of creating a LoFi prototype is to brainstorm, design, create, test, and communicate user interfaces (Snyder, 2003). Hence, LoFi prototypes are ideally created in the initial stages of UI prototyping to ideate and communicate functionalities without consuming much time or effort in development. Such LoFi prototypes can be a wireframe with semantic information (Figure 1.1a) (Brown, 2011), a roughly drawn or a refined sketch that shows visuals, including colors, icons, and controls placement (Figure 1.1b) (Rudd et al., 1996).



(a) A sample LoFi UI wireframe prototype



(b) A sample LoFi sketch prototype

**Figure 1.1:** Sample LoFi prototypes of a UI screen



Typically, UI designers use *paper & pencil*, *tablet & stylus*, and *whiteboard & post-it* for creating such LoFi prototypes (Snyder, 2003). Sketching in paper and pencil is the most preferred LoFi prototyping technique by UI designers (Suleri, Pandian, et al., 2019). Sketching interfaces plays a vital part in ideation and reifying design ideas that Fallman (2003) proclaims that sketching is the "archetypical activity of a designer." Moreover, Buxton (2011) defines sketching as a tool of thought and a fundamental cognitive process of designers that acts as a communication medium between designers.

Another technique for creating LoFi prototypes is to create wireframes using digital tools such as *Balsamiq* (2021), *Moqups* (2021), *NinjaMock* (2021), etc. These wireframes layout the contents of UI design and their properties similar to sketches; however, they are more structured and refined than sketches. A common characteristic of a wireframe is that they use consistent design to denote a UI element, e. g. images are represented as rectangles with crosses (Brown, 2011). Although UI designers generally use wireframes for LoFi prototyping, Snyder (2003) argues whether they must be included in LoFi prototypes and concludes that wireframes without semantic content cannot be considered a LoFi prototype. He concludes that without the semantic information and representing most of the content by rectangles, it is hard to infer whether the designer and the user understand a wireframe similarly (Snyder, 2003).

There are several advantages to creating LoFi prototypes. As Snyder (2003) remarks, it gives "Maximum Feedback for Minimum Effort." He further declares that the LoFi prototype is the most effective means of getting make-it-or-break-it information about the final UI design. Also, among all the prototyping fidelities, designers find LoFi prototyping the quickest and the most inexpensive way to portray a UI design and its functionalities (Duyne et al., 2002). Notably, they play a crucial role in gathering requirements and analyzing the product before development.

Despite these advantages, LoFi prototypes have a few significant disadvantages. By quickly sketching LoFi prototypes, we lose the ability to evaluate the intricate details of a design with users (Rudd et al., 1996; Snyder, 2003; Walker et al., 2002). A study by Walker et al. (2002) shows that users gave significantly more comments and pointed to different types of usability issues when evaluating with higher fidelity prototypes than LoFi prototypes. Therefore, to evaluate the interface with users, a deciding factor in UI design, higher fidelities perform better than the LoFi prototype (Walker et al., 2002).

Nevertheless, higher fidelity prototypes (MeFi and HiFi) sacrifice prototyping speed for an accurate depiction of the final UI design; they are significantly harder to create, modify, and reiterate. Their key advantage over LoFi is that they closely resemble the final system and, therefore, are more suitable than LoFi to evaluate the UI with users and get beneficial feedback.

*In the traditional prototyping process, designers ideate with LoFi prototypes, then manually create and reiterate MeFi and HiFi prototypes to evaluate their system.*

Consequently, UI designers follow the traditional prototyping process of ideating with LoFi prototypes, then manually creating and reiterating MeFi and HiFi prototypes to evaluate their system. This process of providing realism is costly, so Dix (2004) argues that there must be support for the designers to quickly and efficiently create a realistic prototype. Therefore, many researchers attempt to automate the prototyping process or allow designers to generate UI design layouts using machine intelligence.

## 1.2 AUTOMATING UI PROTOTYPING

The desire to automate the design process is not a novel notion. Nearly four decades ago, Mostow (1985) proposed automating design using Artificial Intelligence (AI). He envisioned that we could decrease the cost and improve the reliability of generating multiple designs while enhancing the productivity of human designers using AI. The earliest attempt to automate the prototyping process was by Landay (1996). Landay created SILK, a prototyping tool, to transform a LoFi prototype sketch to code using a pattern recognition algorithm (Landay, 1996). Following his footsteps, many researchers attempted to automate this UI prototyping process using different pattern recognizers. Since the recent advancements in AI, many research projects have further attempted to improve recognizing LoFi prototypes using Deep Neural Networks (DNNs).

*Current research in automating UI design process focuses on automation while disregarding the autonomy of UI designers.*

Similar to automating the design process, a few research attempted to generate UI designs without designers' intervention by using DNNs. Recent advancements in Deep Generative models (I. J. Goodfellow et al., 2014) paved the way to further research in generating UI layouts and wireframes. Recently, LayoutGAN (J. Li et al., 2019) and LayoutTransformer (Gupta et al., 2020) models showed that UI layouts could be learned and replicated by these Deep Generative models.

However, these research projects focus on enhancing AI models to automate the UI design process while disregarding the much-needed autonomy of UI designers. Essentially they attempt to replace designers instead of assisting them during the process.

### 1.3 HUMAN CENTERED AI

Current research in applying AI in UI design disregards the importance of UI designers. Shneiderman (2020b) argues that current AI research must balance automation by appreciating the users' desire to be in control of the technologies. To exemplify his opinion, he proposed a framework for Human-Centered AI (HCAI) to show that even a highly automated system can allow its users to be in control (Shneiderman, 2020a). He concludes that only such an approach would support users' ability, increase their self-efficacy, and enable creativity. Therefore, well-designed automation preserves human control when appropriate while increasing performance and creativity by using AI as assistance.

This thesis explores a human-centred approach to provide AI assistance during the UI design, particularly the LoFi prototyping process. We consider the machine's role in our proposed tools as no different from the apprentices of renaissance art maestros. An apprentice's task is to assist the artist prepare materials and execute the less critical and quite tedious decorative parts of frescoes or statues. Similarly, delegating the tedious and repetitive rework to AI-powered tools while retaining the creative UI design process would greatly benefit designers. It would lead us to a future where AI and humans co-create creative solutions.

*This thesis explores providing AI assistance to UI designers for LoFi prototyping while preserving their autonomy.*

### 1.4 THESIS STATEMENT

In this thesis, we explore the means of providing AI assistance to the UI designers during the LoFi design process while allowing them to remain in control of the process. Subsequently, we evaluate the AI tools for their performance and designer satisfaction.

### 1.5 SCOPE

As UI designs are made for various platforms, we have scoped this thesis to support the smartphone UI designing process. Additionally, following the footsteps of the prior research in automating UI prototyping and capitalizing on the advantages of LoFi prototyping, we further scope this research to assist designers using AI during the LoFi prototyping process of smartphone UIs.

**Scope:** LoFi prototyping for smartphone applications

## 1.6 RESEARCH QUESTIONS

We formalize this thesis by asking the following four questions.

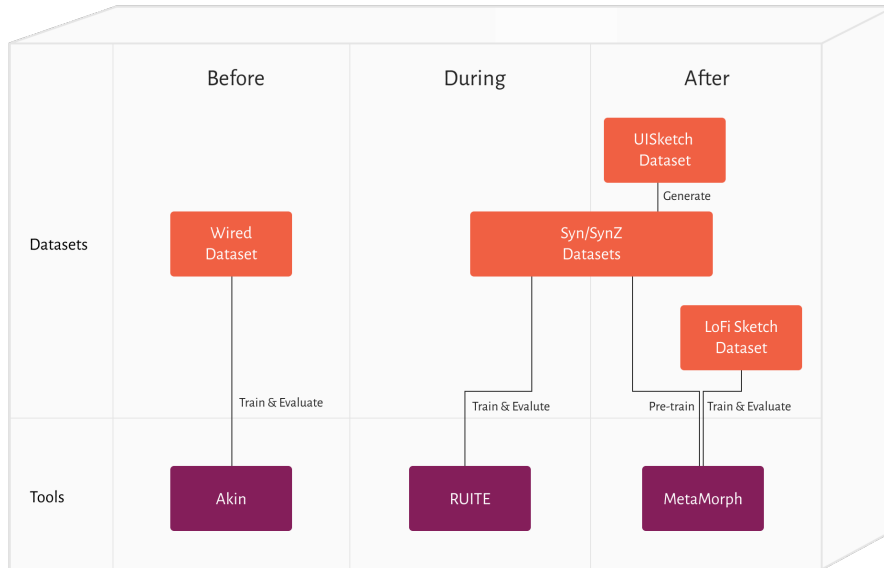
- RQ1** How well do the state-of-the-art deep learning models recognize LoFi sketches?
- RQ2** How do we represent LoFi designs as features for training deep learning models?
- RQ3** How can we provide AI assistance to the LoFi design process?
- RQ4** How does this AI assistance impact designer satisfaction during the LoFi design process?

## 1.7 RESEARCH DESIGN

**Research approach:**  
*Pragmatic worldview  
 with mixed-methods  
 research design*

In this thesis, we followed a pragmatic worldview and used the mixed-methods research design (Creswell et al., 2019). According to this pragmatic approach to research, we conducted quantitative and qualitative methods to answer the research questions.

To address our research questions, we first explored the existing literature and datasets that enable AI assistance in UI design. This literature review identified that the only large-scale dataset available on the UI design domain is the RICO dataset, which contains 72k annotated smartphone UI screenshots. However, a significant limitation of this dataset is that the annotations were generated using automated means; therefore, they are unreliable. If an AI model is trained or evaluated with this dataset, these inaccuracies will propagate to the AI model, thus affecting its quality. Therefore, we aim to collect large-scale datasets ensuring data representation and diversity by collecting and refining them from various sources and participants spread across different countries. To achieve this goal, we accumulated, classified, and refined screenshots from the RICO dataset and real-life smartphone applications. We also collected UI sketches from participants from various countries with different prototyping experiences using paper and digital questionnaires as medium and pencil, pen and stylus as input devices. We, thus, ensure the quality and generalizability of the AI model trained and evaluated with such datasets. With these objectives, we produced four large-scale open-access datasets: *UISketch dataset*, *Syn & SynZ dataset*, *LoFi sketch dataset*, and *Wired dataset*.



**Figure 1.2:** Overview of the datasets and AI tools in the BlackBox toolkit

We used these datasets to train and evaluate our three AI tools that answer the other research questions.

To address RQ1, we started our research by exploring how the existing state-of-the-art AI models understand the LoFi sketches. We trained and evaluated 26 image recognition DNNs with the 17,979 sketches from our UISketch dataset. Further, to compare this AI recognition accuracy with the UI designers, we performed a perceptual study with 76 UI designers and asked them to categorize UI element sketches. These results show that the state-of-the-art image recognition DNNs can recognize UI element sketches (91% accuracy) comparably to UI designers (96% accuracy).

To address RQ2, RQ3, and RQ4, we explored three different aspects to provide AI assistance to UI designers: before, during, and after LoFi prototyping. By addressing these research questions, we created the **BlackBox toolkit**, which contains three AI tools: Akin, RUIE, and MetaMorph. We describe these tools briefly in the next section. [Figure 1.2](#) shows an overview of the datasets and tools in this toolkit.

Specifically, to address RQ2, we investigated how to represent UI design as features for training and evaluating these three AI tools. This investigation concluded that for Akin, representing wireframes as semantically coloured images produced satisfactory results. However, for RUIE and MetaMorph, representing the semantic information of sketches as a tensor of bounding boxes produced the best results.

To address RQ3, as mentioned above, we provided AI assistance before, during, and after the LoFi design process by training three appropriate AI tools.

Finally, to address RQ4, we conducted quantitative and qualitative studies to evaluate each tool individually and understand designer satisfaction while using the AI tool. We applied all three tools individually in their appropriate scenarios and utilized the standard After Scenario Questionnaire (ASQ) (Lewis, 1991a) to measure the designer satisfaction.

## 1.8 RESEARCH CONTRIBUTIONS

Through this research, we contribute four open-access datasets and three open-source artefacts.

### 1.8.1 Datasets

We collected and refined four datasets to train the AI models in the Black-Box toolkit. They are briefly introduced below.

#### *UISketch dataset*

UISketch dataset contains 17,979 hand-drawn sketches of 21 UI element categories collected from 967 participants, including UI/UX designers, front-end developers, HCI, and CS grad students, from 10 different countries.

#### *LoFi sketch dataset*

This dataset contains 4,527 LoFi sketches of more than 100 UI screenshots representing more than ten UI design patterns. We collected these sketches from 361 participants from 76 countries.

#### *Syn & SynZ synthetic LoFi sketch datasets*

Syn and SynZ datasets together contain 300,377 synthetic LoFi sketches with their respective annotation file containing a list of constituent UI elements and their locations in that sketch. We used 17,979 UI element sketches from the UISketch dataset to generate these datasets. The difference between Syn and SynZ datasets lies in the synthetic sketch generation algorithm. While the 125,000 synthetic LoFi sketches in the Syn dataset

are generated by randomly allocating the UI element sketches, the 175,377 synthetic LoFi sketches in the SynZ dataset were statistically generated to resemble real-life UI screenshots. To generate SynZ, we analyzed, enhanced, and extracted annotations from the RICO dataset (Deka et al., 2017).

#### *Wired dataset*

This dataset contains 2,751 images android UI screenshots from the RICO dataset (Deka et al., 2017) classified into 5 UI design patterns. Each UI screenshot in this dataset is annotated into three UI layout layers with 100 different UI element categories and their respective bounding box.

#### 1.8.2 *Artifacts*

In this thesis, we created three AI tools by exploring the AI assistance before, during, and after UI prototyping. We name the collection of these three tools the BlackBox toolkit. These AI tools are briefly described below.

##### *Akin*

Akin is a UI wireframe generator that allows designers to chose a UI design pattern and provides them with multiple UI wireframes for a given UI design pattern. We fine-tuned Self-Attention Generative Adversarial Network (SAGAN) trained with 500 UI wireframes of 5 android UI design patterns to create Akin. Upon evaluation, Akin's generative model provides an Inception Score of 1.63 (SD=0.34) and Fréchet Inception Distance of 297.19. We further conducted user studies with 15 UI/UX designers to evaluate the quality of Akin-generated UI wireframes. The results show that UI designers considered wireframes generated by Akin as good as those made by designers. Moreover, designers identified Akin-generated wireframes as designer-made 50% of the time. Further, the ASQ results indicate that designers experience an above-average satisfaction level by using Akin to generate UI wireframes before their LoFi prototyping process.

*Akin assists designers before their LoFi prototyping task by generating wireframes for a given UI design pattern.*

##### *RUI TE*

RUI TE is a UI layout refiner that optimizes UI wireframe layouts using a Transformer Encoder model. We trained RUI TE by adding noise to misalign 35,072 UI layouts as input and the original aligned layout annotation



*RUIITE provides AI assistance to UI designers during LoFi prototyping by aligning and grouping wireframes to UI layouts.*

as output. Upon evaluation with 500 real-life UI layouts, RUIITE provides an alignment score of 0.87, improvement margin of ~38%, grouping accuracy of 20%, and mean Average Precision of 58.53%. Our user evaluation results indicate that both novice and experienced participants prefer refinement of UI wireframe using AI in a button-click. Further, the ASQ results indicate that designers experience an above-average satisfaction level for using AI assistance during their LoFi prototyping task.

### *MetaMorph*

*MetaMorph assists UI designers after they prototype LoFi sketches by enabling transformation to higher fidelities.*

MetaMorph is a UI element detector that detects the constituent UI elements of LoFi sketches, thereby enabling the transformation of LoFi prototypes to higher fidelities. We used the Syn, SynZ, and LoFi sketch datasets to train MetaMorph. MetaMorph provides 47.8% mAP for hand-drawn LoFi sketches. The ASQ results indicate that designers experience an above-average satisfaction level by utilizing AI assistance to transform LoFi sketches. Also, their qualitative feedback indicates that they perceive utilizing AI as a novel and useful approach to transform LoFi sketches into higher fidelities.

## 1.9 OUTLINE

This thesis is organized into the following parts.

**LITERATURE REVIEW:** In this chapter, we review the existing literature in two aspects. We first explore the existing UI screenshots and UI sketch datasets. Then, we look at the existing research that applies AI in the UI prototyping process.

**BLACKBOX TOOLKIT:** In this chapter, we explain the objective of this thesis and outline the components in the BlackBox toolkit.

**DATASETS:** This part contains four chapters. Each chapter describes the data collection or generation, verification, and refinement procedure of one of the four datasets from our research contribution.

**FORMATIVE ANALYSIS:** In this part, we explain our formative study in which we measured computational accuracy and human recognition accuracy of UI element sketches. Further, we discuss and compare the differences in how well humans can recognize UI element sketches compared to machines' performance.



**AKIN:** This part describes Akin, the UI wireframe generator, in six chapters, explaining the related work, experiments, implementation details, AI model evaluation, user evaluation, and discussion.

**RUIE:** This part describes the RUIE, the UI layout refiner. We explain RUIE in six chapters, describing the background, experiments, implementation details, AI model evaluation, user evaluation, and summary.

**METAMORPH:** In this final part, we describe MetaMorph, a UI element detector. We explain the prior research, experiments, implementation details, AI model evaluation, user studies, and a discussion of MetaMorph in six chapters.

**CONCLUSION:** In the final chapter of this thesis, we summarize the tools in the BlackBox toolkit and discuss the exploration of assisting AI tools in UI design.



---

## LITERATURE REVIEW

---

In this chapter, we explore various research that investigates AI support in UI design. Firstly, we present a comprehensive list of researches in this domain and their research focus. Here, we do not restrict ourselves to a particular prototyping fidelity. Later, we narrow down this list and briefly discuss the datasets and literature that applied AI to support the LoFi prototyping process.

### 2.1 AI SUPPORT IN UI PROTOTYPING

We performed a systematic mapping of the existing literature from the last three decades on AI support in UI design. We conducted this literature review in two stages: identifying all literature through keyword, reference and citation search; and filtering identified literature that fits the UI design process. In the first stage, we searched the commonly published digital databases and libraries such as ACM digital library<sup>1</sup>, IEEE Xplore Digital library<sup>2</sup>, arXiv.org e-print archive<sup>3</sup> for journal articles, conference papers, conference posters, work-in-progress papers, preprint articles, and research articles relevant to the domain. Further, using the connected papers website<sup>4</sup>, we identified a few more relevant research from the references and citations of the identified articles. As the applications of AI in UI design peaked in recent years, we did not exclude the literature based on page length or the number of citations. In the second stage, we reviewed the literature title, abstract, introduction and conclusion for all the identified articles. During this stage, we excluded research that primarily focuses on graphic, magazine or print-media designs. Also, we excluded research that does not apply AI solutions to impact the UI design process.

---

<sup>1</sup> <https://dl.acm.org/>

<sup>2</sup> <https://ieeexplore.ieee.org/>

<sup>3</sup> <https://arxiv.org/>

<sup>4</sup> <https://www.connectedpapers.com/>

**Table 2.1:** Split of literature based on the prototype fidelity, research focus and AI method applied

Prototype Fidelity	Research focus	AI method	No. of existing research	Percentage of existing research
LoFi - Sketch	Automated transformation to Code	Deep Neural Networks	9	23.68%
		Hybrid	2	5.26%
		Pattern Recognition	8	21.05%
LoFi - Wireframe	GUI Similarity Search	Deep Neural Networks	2	5.26%
	Automated transformation to Code	Hybrid	1	2.63%
	UI Layout Refinement, UI Wireframe Generation	Deep Neural Networks	1	2.63%
	UI Wireframe Generation	Deep Neural Networks	2	5.26%
MeFi	Automated transformation to Code	Deep Neural Networks	5	13.16%
		Hybrid	2	5.26%
		Pattern Recognition	2	5.26%
	GUI Similarity Search	Deep Neural Networks	1	2.63%
	UI Design Auto Completion	Deep Neural Networks	1	2.63%
	UI Layout Refinement	Algorithmic	1	2.63%
	UI Wireframe Generation	Deep Neural Networks	1	2.63%
Grand Total			38	100.00%

Through these steps, we identified 38 research projects: 32 peer-reviewed academic research, one preprint, one commercial product, and four proof-of-concept research projects. We categorized them into their targeted platform, research focus, prototyping fidelity focus, and AI method. We summarize them in [Table 2.1](#), [Table 2.2](#), and [Appendix a](#).

*These research projects focus on three major research areas: converting LoFi prototypes to code, generating UI wireframes, and searching for UI screenshots similar to the LoFi sketches.*

Out of the 38 research projects, 29 projects concentrate on automating the transformation of prototypes. Specifically, 19 of these 29 research studies concentrate on converting the LoFi prototype directly to code, and the rest ten projects explore converting the MeFi prototype to code. The subsequent most popular research focus on this domain is on UI wireframe generation conducted by four academic research projects. All four of these research utilize DNNs to generate UI wireframes for the smartphone platform. One of these four research projects attempts to refine a generated UI layout by aligning and grouping them automatically. Another research also attempts to refine web UI layouts based on user constraints using an algorithmic approach as an alternative. Apart from these topics, three research projects explore GUI similarity search, employing DNNs to search for UIs similar to the user's selection. One unique research attempts to auto-complete UI wireframe while a designer designs a UI design.

**Table 2.2:** Comprehensive list of research on AI support for UI prototyping

Research Work	Type	Platform	Research focus	Prototype Fidelity focus	AI method
Landay and Myers, 1995	Academic Research	Desktop	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Caetano et al., 2002	Academic Research	Desktop	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Plimmer et al., 2003	Academic Research	Desktop	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Coyette et al., 2004	Academic Research	Desktop	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Segura et al., 2012	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Halbe et al., 2015	Academic Research	Smartphone	Automated transformation to Code	LoFi - Wireframe	Hybrid
R. Huang et al., 2016	Academic Research	Web	Automated transformation to Code	MeFi	Pattern Recognition
Benjamin, 2017	Proof-of-concept Project	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
S.-H. Li et al., 2017	Academic Research	Smartphone	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Beltramelli, 2017	Academic Research	Smartphone, Web	Automated transformation to Code	MeFi	Deep Neural Networks
Han et al., 2018	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Kim et al., 2018	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Microsoft AI Labs, 2018	Proof-of-concept Project	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Park et al., 2018	Academic Research	Smartphone	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Ashwin, 2018	Proof-of-concept Project	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Bajammal et al., 2018	Academic Research	Web	Automated transformation to Code	MeFi	Pattern Recognition
C. Chen et al., 2018	Academic Research	Smartphone	Automated transformation to Code	MeFi	Deep Neural Networks
Y. Liu et al., 2018	Academic Research	Smartphone	Automated transformation to Code	MeFi	Deep Neural Networks
Moran et al., 2018	Academic Research	Smartphone	Automated transformation to Code	MeFi	Hybrid

Table 2.2 continued from previous page

Research Work	Type	Platform	Research focus	Prototype Fidelity focus	AI method
Wallner, 2018	Proof-of-concept Project	Smartphone	Automated transformation to Code	MeFi	Deep Neural Networks
Aşıroğlu et al., 2019	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Hybrid
Jain et al., 2019	Preprint	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Narendra et al., 2019	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Yun et al., 2019	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
Uizard, 2019	Commercial	Smartphone	Automated transformation to Code	LoFi - Sketch	Deep Neural Networks
S. Chen et al., 2019	Academic Research	Smartphone	Automated transformation to Code	MeFi	Hybrid
Mistry et al., 2020	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Hybrid
Wimmer et al., 2020	Academic Research	Web	Automated transformation to Code	LoFi - Sketch	Pattern Recognition
Sharma et al., 2020	Academic Research	Web	Automated transformation to Code	MeFi	Deep Neural Networks
Ge, 2019	Academic Research	Smartphone	GUI Similarity Search	LoFi - Sketch	Deep Neural Networks
F. Huang et al., 2019	Academic Research	Smartphone	GUI Similarity Search	LoFi - Sketch	Deep Neural Networks
Manandhar et al., 2020	Academic Research	Smartphone	GUI Similarity Search	MeFi	Deep Neural Networks
Y. Li et al., 2020	Academic Research	Smartphone	UI Design Auto Completion	MeFi	Deep Neural Networks
Dayama et al., 2020	Academic Research	Web	UI Layout Refinement	MeFi	Algorithmic
Lee et al., 2020	Academic Research	Smartphone	UI Wireframe Generation and Refinement	LoFi - Wireframe	Deep Neural Networks
J. Li et al., 2019	Academic Research	Smartphone	UI Wireframe Generation	LoFi - Wireframe	Deep Neural Networks
Gupta et al., 2020	Preprint	Smartphone	UI Wireframe Generation	LoFi - Wireframe	Deep Neural Networks
Zhao et al., 2021	Academic Research	Smartphone	UI Wireframe Generation	MeFi	Deep Neural Networks

The earliest research in providing AI support in UI design is by Landay and Myers (1995). They attempted to convert LoFi sketch to code using a pattern recognizer. Since then, ten more projects have attempted to utilize pattern recognition to provide AI support in UI design. Since 2015, there is a trend in utilizing DNNs for AI support in UI design; 22 research projects used only DNNs, and 5 used a hybrid of pattern recognition and DNNs for research in this domain.

Most of these research projects concentrate on providing support for LoFi prototypes: 21 for LoFi sketches and 4 for LoFi UI wireframes. The rest of the research (14) provides support for MeFi prototypes. Most of these research projects concentrate on either smartphone or web UI designs (17 research works each). Besides four research projects in the early 2000s, no recent project concentrates on AI support of UI design in desktop platforms.

This thesis aims to provide AI support for LoFi prototyping; hence, in the rest of this chapter, we discuss the literature that focused their research on supporting LoFi sketches and wireframes. In the next section, we briefly discuss the UI datasets that enabled these research works, and we sectioned the subsequent sections based on the existing literature's research focus.

## 2.2 UI DATASETS

In this section, we discuss the widely used open-source datasets in the literature discussed above. We list the datasets discussed in this section and their attributes in table 2.3.

### 2.2.1 RICO dataset

The largest repository of smartphone UI screenshots is the RICO dataset collected by Deka et al. (2017) using crowd-sourcing platforms. The RICO dataset contains more than 72,000 Android smartphone screenshots and

*Recent research projects tend to use DNNs or a hybrid of DNN with pattern recognition algorithms.*

*Most of the research focuses on LoFi prototypes of smartphone or web application*

**Table 2.3:** List of UI datasets and their attributes

Dataset	Type	Count	Annotations
RICO dataset	LoFi UI wireframes and UI screenshots	72,000	25 categories of UI elements
Microsoft AI lab dataset	LoFi sketch of Web UI	149	8 categories of UI elements
SWIRE dataset	LoFi sketch of Smartphone UI	3,802	None

their respective Android view hierarchies of 27 app categories. The Android view hierarchy of a UI screenshot specifies the location, category, and layout of UI elements present in the corresponding UI screenshot. This annotation provides rich information of UI elements present in a UI screenshot that enables the dataset to be used for both LoFi UI wireframes and MeFi prototypes.

*RICO is the largest dataset in this domain, with 72k annotated smartphone UI screenshots. A significant limitation of this dataset is that the annotations were generated using automated means; therefore, they are unreliable.*

RICO dataset was further extended by T. F. Liu et al. (2018) by adding semantic annotations for the UI screenshots. In their research, T. F. Liu et al. (2018) categorized the UI elements present in the RICO dataset into 25 categories. For this categorization, they used both manual and automated methodologies. They further classified these UI element categories into 197 text button concepts and 135 icon classes. One drawback of this dataset is that as the UI elements were categorized and annotated using automated means, the annotations are unreliable. Therefore, using this dataset for training DNNs would cascade the inaccuracies to the trained DNN model. However, this dataset is used in many existing works of literature after few refinements.

Recently, Leiva et al. (2020) revised a subset of the RICO dataset to overcome its shortcomings and provided a refined dataset, Enrico (shorthand of Enhanced Rico). This dataset is annotated with human supervision and contains 1,460 UIs classified into 20 design topics. Enrico is annotated into 25 categories of UI elements similar to RICO.

### 2.2.2 Microsoft AI lab dataset

*Microsoft AI lab dataset contains 149 web LoFi sketches annotated with eight categories of UI elements.*

This LoFi sketch dataset was created by Microsoft AI Labs (2018) to train their proof-of-concept project, Sketch2Code. This dataset contains 149 sketches of web UIs annotated with ten categories of UI elements. Although this dataset is used in 3 research projects, it is insufficient to train a generalizable DNN model due to the small size. Also, the collection method of this dataset is not specified to understand the characteristics of this dataset.

### 2.2.3 SWIRE dataset

*SWIRE is an unlabelled, unannotated dataset of 3,802 smartphone lo-fi sketches drawn by five UI/UX designers.*

F. Huang et al. (2019) provided the SWIRE dataset containing 3,802 lo-fi sketches based on 2,201 UI screenshots from the RICO dataset to train their GUI similarity search DNN. SWIRE is an unlabelled, unannotated dataset of smartphone lo-fi sketches. These sketches were collected from



four UI designers. Although this is the most extensive collection of the LoFi sketch dataset in the existing literature, this is not yet used in any other research other than GUI similarity search.

The following section discusses the research works that used these datasets to create AI support in UI design.

### 2.3 AUTOMATED TRANSFORMATION OF PROTOTYPE FIDELITY

Since Landay and Myers (1995) proposed Sketching Interfaces Like Krazy (SILK) tool in 1995, many research projects endeavoured to generate code from LoFi sketches and wireframes. This section describes these research studies and the sketch recognition techniques used by them. We summarize these research in table a.1 with the dataset used, model details, and evaluation results.

#### 2.3.1 Pattern Recognition

All the early research works conducted by Caetano et al. (2002), Coyette et al. (2004), Landay (1996), and Plimmer et al. (2003) adapted classical pattern recognition algorithms like Rubine recognizer by Rubine (1991) or CALI recognizer by Fonseca et al. (2002) to transform lo-fi sketches to Java, Visual Basic, and UsiXML code. As an alternative approach, Bajammal et al. (2018), R. Huang et al. (2016), S.-H. Li et al. (2017), Narendra et al. (2019), Segura et al. (2012), and Wimmer et al. (2020) created custom recognizer using convex hull detection, unsupervised clustering, retraining DoodleClassifier to identify certain image features, analyze these features, and detect the constituent elements.

*Most of the earlier pioneering works on this domain used pattern recognition to convert LoFi sketches to code.*

#### 2.3.2 Deep Neural Networks

Benjamin (2017) from the Airbnb design team attempted a different approach to solve this research problem by employing DNNs instead of pattern-recognition algorithms to generate code from LoFi sketches. Another such attempt was made by the company Uizard (2019) to create their eponymous commercial product. However, both these projects revealed neither their architecture nor their evaluation results.

In the following year, few proof-of-concept projects such as Sketch-code by Ashwin (2018) and Microsoft AI Labs (2018) released open-source

*Almost all the recent research in this domain use DNNs to convert LoFi sketches to code. However, all the DNNs were trained with small-scale datasets, thus affecting its generalizability.*

projects to convert LoFi sketches to code. Ashwin used image captioning DNN to generate HTML markup from LoFi sketches of web UI. He used a synthetic dataset of 1,500 sketches, generated by manipulating CSS code to resemble freehand sketches, to train their model. Microsoft AI Labs used their custom vision service to identify the most common HTML elements like button, textbox, and image. Besides, it can also detect handwritten text and layout information using Microsoft computer vision service. However, Microsoft's system is trained with a tiny dataset of 149 sketches and focuses only on the ten most common HTML elements. Both these proof-of-concept projects provided their code and dataset as open-source but did not provide a machine or human evaluation to judge their system's performance.

Besides these commercial and proof-of-concept research projects, few academic projects, by Han et al. (2018), Kim et al. (2018), Park et al. (2018), and Yun et al. (2019), created DNN based UI element detectors. However, these projects were published as work-in-progress, and most of them provided neither their ground-truth dataset nor their evaluation results. Only Kim et al. (2018) claims that their model performs with 91% Precision and 86% recall without any further information on the evaluation.

To conclude this section, the research projects discussed solely focus on enhancing AI models to automate the UI design process while disregarding UI designers' much-needed autonomy. All the projects directly convert LoFi sketch to code and do not provide an intermediate step to allow designers to customize the detected UI elements. Essentially they attempt to replace designers instead of assisting them during the process.

Besides these 29 research works on converting LoFi sketches to code, a few existing research projects attempt to generate and refine UI wireframe layouts. We will discuss them briefly in the next section.

## 2.4 UI WIREFRAME GENERATION AND REFINEMENT

Apart from automatic fidelity transformation, three research projects attempt to generate UI wireframes. These projects expand on the prior research in automatic generation of generating document or magazine layouts such as design scape. All the research projects that generate and refine UI wireframes use either DNNs or a hybrid of DNNs and classic pattern recognition algorithms. Most of these projects use Generative Adversarial Networks (GAN) for this task. GANs learn complex distributions and use them to synthesize semantically meaningful results in multiple

domains such as image, text, audio, or spatial. A growing body of research on Image synthesis uses GANs, and a subset of it is applied in UI wireframe generation. In this section, we summarize these three research projects. Also, we list these research in table a.2 with the dataset used, model details, and evaluation results.

#### 2.4.1 LayoutGAN

J. Li et al. (2019) introduced LayoutGAN that generates graphic, document and UI wireframe layouts by modelling geometric relations of different types of 2D elements (such as UI elements) present in the content. The advantage of LayoutGAN is that it is not limited to any particular domain and can synthesize any set of 2D graphical elements in any given design domain, provided the DNN is fine-tuned further. LayoutGAN is constructed with the traditional GAN architecture. It contains a generator that takes a set of randomly placed 2D graphic elements as input and utilizes self-attention layers to refine their position to produce a realistic layout. The best performing LayoutGAN model uses a wireframe rendering discriminator, which renders the wireframes from input labels and class probabilities to compare whether it is realistic or not. The authors used the RICO dataset with 5 UI elements to show the use-case of LayoutGAN. However, they did not conduct any human or machine evaluation of the generated UI wireframes.

*LayoutGAN uses a modified GAN model to generate UI wireframes. However, the authors do not provide any evaluation to measure the quality of generated wireframes.*

#### 2.4.2 LayoutTransformer

Gupta et al. (2020) proposed another architecture using an auto-regressive self-attention network called LayoutTransformer to generate document and UI wireframes. In contrast with LayoutGAN, this model can take an empty or partial layout as input and generate realistic document or UI wireframe layouts. This ability is possible due to the way a wireframe layout is represented by LayoutTransformer architecture—a combination of category, position and size in a discrete multinomial distribution. This novel representation of UI layout combined with the power of the auto-regressive model allows the model to learn the complex relationship between each element in the layout. The LayoutTransformer model is trained with a subset of the RICO dataset. It provides 33.6 Coverage 23.7 overlap in spatial distribution analysis. However, similar to LayoutGAN, it is not evaluated with users.

*LayoutTransformer uses Transformer models to generate document and UI wireframes. The generated wireframes were evaluated quantitatively but were not evaluated with designers.*

### 2.4.3 Neural Design Network

*Neural Design Network generates UI wireframes and refines the alignment in the generated wireframe. Similar to other projects, the generated UI wireframes were not evaluated by designers.*

The latest academic research on UI wireframe generation is the Neural Design Network by Lee et al. (2020). Unlike the other two research mentioned above, the authors also refine the generated wireframes. The most exciting approach taken by the authors is to represent UI layouts as graphs. Such a layout graph is constructed by representing UI elements as nodes and their relationship (location above, below) as edges. The Neural Design Network is constructed using a Graph Convolution Network (GCN) that generates UI wireframes as layout graphs based on user constraints. However, as such, generated layout graph can be misaligned. Therefore, to improve the generated layout's alignment, the authors create another GCN to refine these layouts. This GCN is trained using a subset of the RICO dataset and provides a Frèchet Inception Distance (FID) score of  $143.51 \pm 22.36$  upon evaluation. Similar to both the research mentioned above, this system as well is not evaluated with users.

In summary, the major shortcoming of all these research projects is that they have not been evaluated by designers to uncover their needs and desires. Also, these research projects do not consider that not all UI wireframes fit any use case. Therefore the generated designs from such DNNs will be generic and cannot be used for a specific scenario.

In the upcoming section, we look at few use cases that attempt to provide designers with a quick search for inspirational examples.

## 2.5 GUI SIMILARITY SEARCH

This section briefly discusses the two projects on sketch-based UI screenshot retrieval. We summarize these research in table a.3 with the dataset used, model details, and evaluation results.

*Two research projects explored LoFi sketch-based GUI screenshots retrieval and claimed that the inspirational UI screenshot search with sketching to be helpful to designers.*

Ge (2019) and F. Huang et al. (2019) utilize DNNs to reduce a LoFi sketch and compare it with UI screenshots to create a sketch based GUI similarity search engine. Ge (2019) declare they use the network created by S. Chen et al. (2019) to do this task. However, they do not provide their model evaluation or user evaluation to justify their model's performance.

F. Huang et al. (2019) created a dataset of 3,802 images based on 2,201 UI screenshots from the RICO dataset to create SWIRE. The authors use a modified VGG-A network to create the SWIRE model and use a triplet loss function to train it. This model provides 15.6% top-1 accuracy and 60.9% top-10 accuracy in fetching similar GUIs. The authors further conduct a

qualitative study with five designers. From their feedback, the authors claim that the designers were satisfied with SWIRE and found the inspirational UI screenshot search with sketching helpful.

## 2.6 IDENTIFIED RESEARCH GAPS

From our literature review, we identified that the existing research on AI support for UI design provides adequate evidence that current state-of-the-art AI models can efficiently assist UI designers in their design process, and AI support in UI design is beneficial for designers. However, most of the research projects were not evaluated with UI designers to understand their needs and desires. Besides, all of these research projects focus on enhancing automation using AI models while disregarding the autonomy of UI designers.

The significant limitations in the current research are that the automated prototype fidelity transformation research does not provide an intermediate step to allow designers to customize the detected UI elements, and the research on UI wireframe generation does not generate wireframes based on designer specified use cases. This thesis aims to bridge this research gap by conducting a systematic analysis of AI support in UI design.

*Most of the research projects were not evaluated by UI designers. All of these research projects focus on enhancing automation using AI models while disregarding the autonomy of UI designers.*

## 2.7 SUMMARY

This chapter summarized the existing research on AI support for LoFi prototyping conducted in the past three decades by academic, commercial, and proof-of-concept research projects. These research projects focus on three major research areas: converting LoFi prototypes to code, generating UI wireframes, and searching for UI screenshots similar to the LoFi sketches. Through our literature review, we identified a need to move the focus of this research domain from AI-centric research to Human-Centric AI investigations. In this thesis, we aim to bridge these research gaps by conducting systematic research on the AI support of the LoFi prototyping process. In the upcoming chapter, we explain our objectives and approach towards this research.



---

## BLACKBOX TOOLKIT

---

This chapter briefly explains the primary goal of creating the BlackBox toolkit and the interplay of datasets and tools inside this toolkit.

### 3.1 GOALS

The key objective of this thesis is to explore the means of providing AI assistance to UI designers throughout the LoFi design process without sacrificing their autonomy. Consequently, through this thesis, we aim to create different datasets and AI tools to gain insight into the LoFi design process in three stages: before, during, and after LoFi prototyping; thus, fostering further research in this domain. In addition to the overarching goal, we formulate further goals for each component type (dataset and AI tool) in the BlackBox Toolkit.

The primary goal of each dataset in the toolkit is to create a diverse collection of data spanning participants from different countries, experience in prototyping, and input medium. Further, we aim to provide these datasets as open access to further research without scoping or limiting their applications. Similarly, for each AI tool, we primarily aim to create them as pluggable, generalisable and retrainable systems. Also, we intend to provide these AI tools as open-source applications that can be fine-tuned and attached to any modular application. Moreover, by evaluating these AI tools with UI designers, we aim to systematically analyse the satisfaction and interest of novice and experienced users in using AI as assistance during the UI design process. We briefly introduce the Blackbox toolkit in the next section.

### 3.2 BLACKBOX TOOLKIT

Blackbox toolkit is a collection of four datasets and three AI tools created to assist UI designers throughout the LoFi prototyping process, augmenting their workflow without sacrificing autonomy. We named this toolkit "Blackbox" as a user of this toolkit should be able to use the tools in it as any other design tool, such as a pencil or compass, without worrying about neither the manufacturing process nor the inner workings of the tool itself. We ourselves use numerous tools from simple door handles to complex tools such as cars to achieve our goals, which exist as a "Blackbox" in our daily lives.

The datasets and AI tools in the Blackbox toolkit are readily available for the consumption of UI designers. We have distributed the four datasets as open-access datasets in Kaggle, a public data platform provided by Google. The AI tools are deployed as open-source and open-access Web APIs and plugins for Adobe XD prototyping platform. In the upcoming sections, we briefly explain the rationale for creating the different components in the BlackBox Toolkit.

### 3.3 DATASETS

This toolkit contains four datasets: **UISketch dataset**, ~18k UI element sketches; **Syn & SynZ datasets**, ~300k synthetic LoFi sketches; **LoFi sketch**, ~4.5k real-life LoFi sketches and **Wired dataset**, ~2.7k semantically annotated UI screenshots. Unlike the AI tools, the datasets are not restricted to the different stages of prototyping. Instead, each of these datasets targets one of the two types of LoFi prototypes: LoFi sketches and LoFi wireframes. UISketch, Syn datasets only concentrate on LoFi sketches. Similarly, SynZ and LoFi datasets primarily concentrate on LoFi sketches, but as they contain annotations in addition to sketches, they extend support to LoFi wireframes. Unlike others, the Wired dataset annotations only focus on LoFi wireframes. Nevertheless, beyond the scope of this thesis, SynZ annotations and Wired dataset can also be used in understanding MeFi design and HiFi designs as they are accumulated from and contains UI screenshots.

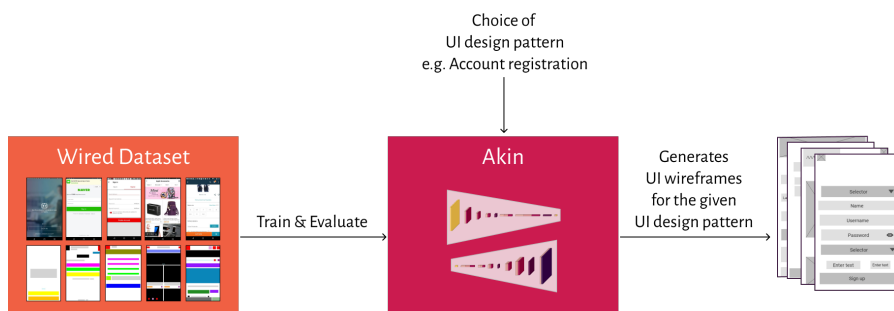


### 3.4 AI TOOLS

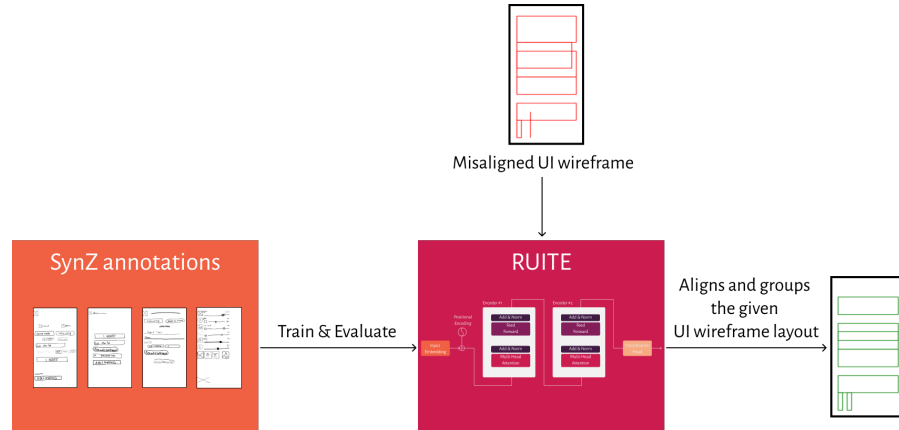
The AI tools use a subset or a combination of the datasets mentioned above for training and evaluation. However, unlike the datasets, the AI tools are conceived to target a specific stage of LoFi prototyping. Furthermore, we use the user evaluation of these AI tools to foster discussion on the satisfaction of designers in utilising AI assistance for LoFi prototyping. In the upcoming subsections, we discuss each stage of LoFi prototyping and the interplay of datasets and tools that target that stage.

#### 3.4.1 Before LoFi prototyping

Before LoFi prototyping, designers look for inspirations or examples by browsing UI design patterns and UI screenshot libraries and use them as a base for prototyping their UI (Herring et al., 2009; Suleri, Pandian, et al., 2019). This thesis explores an alternative approach to designers by proposing a UI wireframe generator that learns from the existing UI design pattern screenshots and generates UI wireframes for a given UI design pattern, thereby providing designers with a base wireframe before starting LoFi prototyping. Also, we aim to create these generated wireframes to be editable so that the designers can modify them and further the LoFi wireframes for their use case. To achieve this, we used a subset of the Wired dataset, which contains semantically annotated UI screenshots, and created Akin, a UI wireframe generator. Figure 3.1 shows the components of the BlackBox toolkit we used to understand this stage of LoFi prototyping.



**Figure 3.1:** Akin assists UI designers before they start prototyping by generating UI wireframes for the chosen UI design pattern. It is trained and evaluated with the Wired dataset



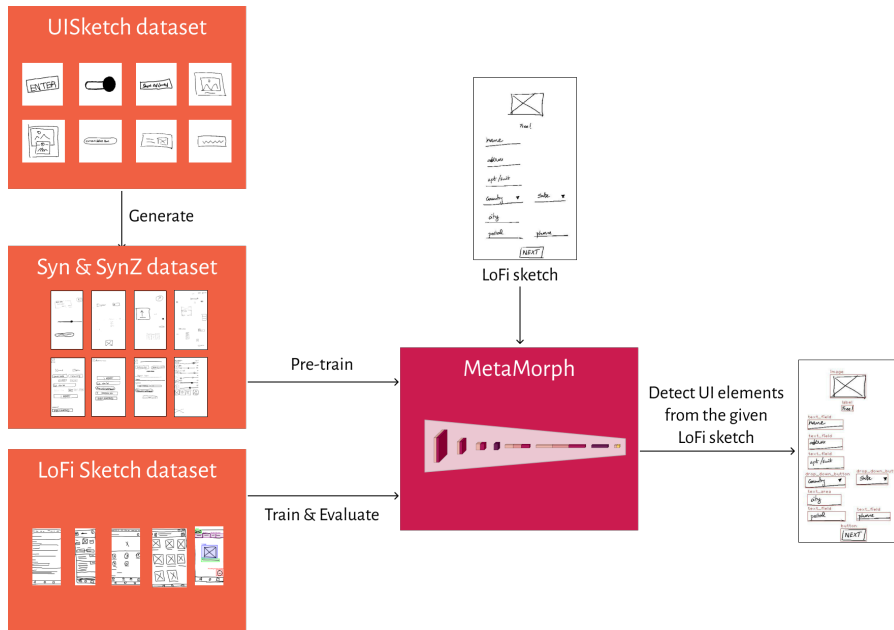
**Figure 3.2:** RUIITE assists UI designers during prototyping by aligning and grouping the UI wireframe layout. It is trained and evaluated with the SynZ dataset annotations

### 3.4.2 During LoFi prototyping

During LoFi prototyping, designers concentrate on ideation and quickly reifying their design concepts (Rudd et al., 1996). Also, they do not heed the aesthetic beautification of the system during LoFi sketching or wireframing. Therefore, to aid designers during their prototyping process, we propose to create an AI tool that quickly aligns and groups the UI elements in a UI wireframe. Thus automatically generating an aesthetically pleasing UI wireframe and assisting UI designers during their prototyping process. To develop this solution, we used the SynZ annotations from our toolkit for training and evaluation and created RUIITE, a UI wireframe refiner. Figure 3.2 shows the components of the BlackBox toolkit we used to investigate this stage of LoFi prototyping systematically.

### 3.4.3 After LoFi prototyping

During the LoFi prototyping phase, designers reiterate and rework multiple LoFi prototypes to finalise a UI design. Then, they convert this LoFi prototype to higher fidelities for evaluation and handover to development. Despite widespread usage, this traditional prototyping process is tedious and time-consuming (Suleri, Pandian, et al., 2019). It limits the number of LoFi prototype ideas that propagate to higher fidelity. To tackle this problem, we propose a UI element detector that can locate and classify UI elements present in a LoFi sketch, thus enabling an automated transformation of LoFi sketches to higher fidelity. To achieve this, we trained an



**Figure 3.3:** MetaMorph assists UI designers after the LoFi prototyping by detecting the constituent UI elements thereby enabling transformation to higher fidelities. This tool is pretrained with the synthetic LoFi sketches from the Syn and SynZ datasets. It is further fine-tuned and evaluated with the LoFi Sketch dataset

object detection model and created MetaMorph, a UI element detector. As an object detection model is data-demanding, we require large-scale datasets to train such a system. Therefore, we used Syn and SynZ datasets, containing ~300k synthetic lofi sketches generated using the UI element sketches in the UISketch dataset. For further training and evaluation, we used the LoFi sketch dataset from our toolkit. Figure 3.3 shows the datasets we used and the tools we created in the BlackBox Toolkit to systematically analyse this stage of prototyping.

By collecting, refining, and creating these datasets and tools in the BlackBox Toolkit, we study the impact of utilising AI assistance in the UI design process. In the upcoming parts of the thesis, we explain each component of the BlackBox Toolkit in detail.



## PART I

# DATASETS

Datasets are fundamental to foster an Artificial Intelligence model. From our literature review, we identified that there is a need for large-scale LoFi sketch datasets to train AI models to support UI design. To bridge this gap, we collected and refined four datasets: UISketch dataset, LoFi sketch dataset, Syn & SynZ datasets, and Wired dataset. Each dataset targets a specific application domain and enables us to train different AI models to support UI design. This part is split into four chapters. In each chapter, we describe the data collection or generation, verification, and refinement procedure of one of the four datasets from our research contribution.



<https://blackbox-toolkit.com/data/>



---

## UISKETCH DATASET

---

In this chapter, we present UISketch dataset, the first large-scale dataset of 17,979 hand-drawn sketches of 21 UI element categories collected from 967 participants, including UI/UX designers, front-end developers, HCI, and CS grad students, from 10 different countries. To create this dataset, we identified a list of UI element categories that needed to be included in the dataset, then we created questionnaires and collected the respective sketches from participants. We then extracted, processed, and validated the collected sketches to create the UISketch dataset. The following sections briefly explain the steps mentioned above. This chapter was published as a part of our research paper, “UISketch dataset” (Pandian, Suleri, and Jarke, 2021b).

### 4.1 OBJECTIVE

Our literature review identified that several research projects attempt to apply AI to improve the traditional LoFi prototyping process (Chapter 2). However, these research projects, such as Microsoft AI Labs (2018), utilize smaller datasets collected from few participants. Using such datasets to train AI models affects their generalizability and overall quality. Also, using such datasets for evaluation does not provide ecological validity to the results. Therefore, the primary objective of collecting the UI sketch dataset is to obtain a large-scale dataset of UI element sketches from a diverse set of participants and different input mediums. Through this dataset, we aim to benchmark and contrast the human accuracy and current state-of-the-art UI sketch classification accuracy in identifying UI element sketches. Moreover, this dataset enables the creation of large-scale synthetic LoFi sketches by stitching the UI element sketches. Also, it can be used in Sketch-based UI element retrieval and automatic sketch completion research.

## 4.2 TAXONOMY OF UI ELEMENT SKETCHES

Before we began the data collection process, we aimed to identify the most common UI elements used by designers during the UI design process. As the design of UI elements differs for different platforms and operating systems (OS), we scoped this research for smartphones, specifically Android OS. To identify commonly used UI elements on Android smartphones, we first determined the widely used design languages for smartphone applications, such as Material Design by Google, 2021, Fluent Design by Microsoft, 2021, and other UI component libraries (Framework7, 2020; GeekyAnts, 2020; React Native Elements, 2020; Shoutem, 2020). We also extracted a list of the most commonly found UI elements from the RICO dataset of Android UI screenshots identified by T. F. Liu et al., 2018. We then excluded the duplicate entries as the nomenclature of similar UI elements differs across different design languages (e.g., dropdown vs select). We further reviewed this UI elements list with five UI/UX designers and altered the list based on their input. Finally, we arrived at a final list of 21 most commonly used UI elements.

*We identified 21 categories of UI elements commonly used in smartphone UI designs: 14 standalone and 7 composite UI elements.*

We further divided this list into composite and standalone UI elements. Standalone UI elements are essential building blocks of UI, such as a text field, button, or checkbox; whereas, composite UI elements consist of one or more standalone UI elements such as a card, grid list, or alert. In total, we chose the most commonly used 21 UI elements: 14 standalone and 7 composite (Table 4.2).

## 4.3 DATA COLLECTION QUESTIONNAIRES

As a next step, we reviewed the existing sketch datasets such as the TU Berlin sketch dataset of everyday objects by Eitz et al. (2012), Sketchy database of sketch-photo pairs by Sangkloy et al. (2016), and SWIRE dataset by F. Huang et al. (2019) of hand-drawn lo-fi sketches to analyze their data collection practices.

These prior data collection studies provided a sample image to the participant corresponding to each object category. Following the data collection practices of F. Huang et al. (2019), we provided sample UI element images and asked participants to sketch the given UI elements in their own way. Additionally, we requested participants not to trace the sample images. As a precaution, we also randomized the sample images (out of 2,400 different sample images of 21 categories of UI elements) provided to



participants to ensure generalized data collected even if they were tracing. Therefore, for each UI element category, we had around 110 sample images. So during our data collection study, each sample image of a particular UI element was repeated for at least 9 participants.

We aimed to collect a comprehensive dataset that includes UI elements sketches hand-drawn on paper and digitally drawn using a stylus. Therefore, we collected UI element sketches using both paper<sup>1</sup> and digital<sup>2</sup> questionnaires.

We designed the paper questionnaire<sup>1</sup> such that each question has an example of a UI element on the left and a rectangular area to sketch the UI element on the right-hand side (Figure 4.1a). These example images were randomly selected and intended to assist the participants with limited awareness of UI elements.

With a design similar to the paper questionnaire, we developed a web application<sup>2</sup>, which only allows stylus input to collect digital sketches of UI elements (Figure 4.1b). The web application starts by requesting participants to provide informed consent. Once the participant agrees to the consent form, the web application creates a secure key for authentication and stores it in their browser's internal database. This step restricts the participants from taking part in the study multiple times. It also helps in restoring the web application from the point they left off in case of internet disconnection.

We also developed a cross-platform iOS and Android application using react-native with similar features as our web application to collect UI element sketches drawn only using a stylus on an iPad and Android tablet.

*We collected sketches using both paper and digital questionnaire (for both desktop and tablet) where participants could sketch UI elements using pen, pencil, or stylus.*

#### 4.4 PILOT STUDY & DESIGN DECISIONS

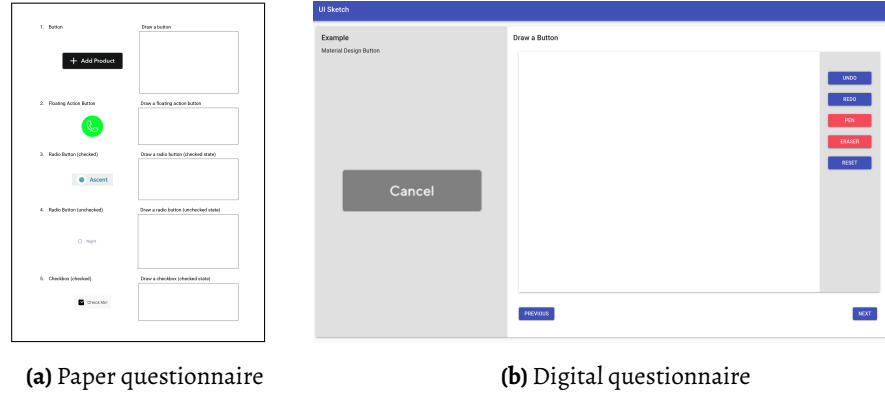
Before we started the mass collection of UI element sketches, we first started with a pilot study of 16 participants of each category: 4 UI/UX designers, 4 front-end developers, 4 HCI grads, and 4 CS grad students. We presented 8 participants (2 of each category) with paper questionnaires and the remaining 8 participants with the digital questionnaires and asked them to sketch all 21 categories of UI elements.

From this study, we understood that asking participants to sketch text and image content independently is unnecessary and redundant—as these UI elements are a part of other standalone UI elements, e.g., button, text

*Our pilot study showed that collecting text and image content independently is unnecessary and redundant; instead, they can be cropped from other UI element sketches to get generalizable data.*

<sup>1</sup> [https://blackbox-toolkit.com/f/uisketch/paper\\_questionnaire.pdf](https://blackbox-toolkit.com/f/uisketch/paper_questionnaire.pdf)

<sup>2</sup> <https://uisketch.web.app/>



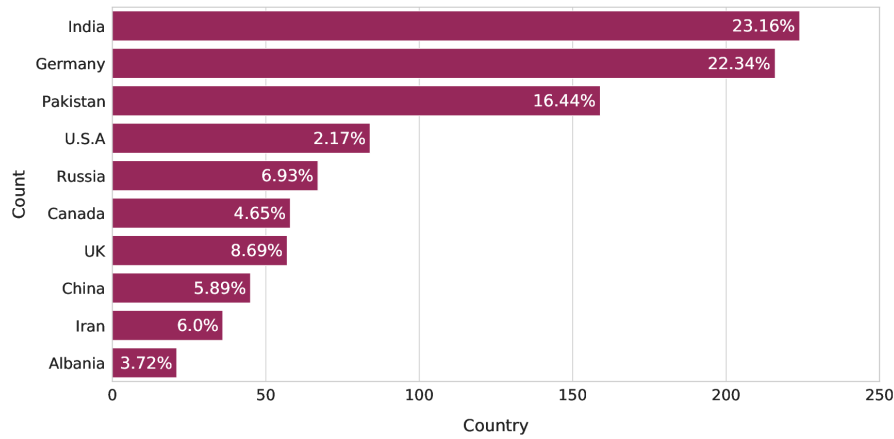
**Figure 4.1:** Paper and digital questionnaire used for collecting UI Sketch dataset. Both questionnaires have an example of a UI element on the left and a rectangular area to sketch the UI element on the right-hand side.

field, text area, and composite UI elements, e.g., grid list, data table, chip. Therefore, we removed text (label) and image UI elements from the paper and digital questionnaires. We later cropped them from the other standalone and composite UI element sketches to obtain generalizable data.

#### 4.5 PARTICIPANTS

*We collected the UI element sketches using paper and digital questionnaires from 967 participants, including UI/UX designers, front-end developers, HCI, and CS grad students, from 10 different countries.*

Over a span of 3 years, we collected UI element sketches from 967 participants: 151 UI/UX designers with 2-14 years ( $M=6.95$ ,  $SD=3.5$ ) of prior prototyping experience, 136 front-end developers with 2-10 years ( $M=5.6$ ,  $SD=2.3$ ) of former UI development experience, 288 HCI graduate students with up to one year of prototyping experience, and 392 Computer Science graduate students who had attended the UI design and development related courses. We did not restrict data collection based on the age or gender of the participants. We collected data from participants belonging to 10 countries: Germany, Russia, China, Albania, India, Pakistan, United Kingdom, Canada, Iran and the U.S.A (Figure 4.2). We used purposive and snowball sampling to recruit these participants. For gathering such a massive amount of participants, we contacted several UI/UX professionals and developer through social media and online platforms. We also reached out to relevant people through multiple universities, meetups, and conferences.



**Figure 4.2:** Demographics of participants as percentage of participants per country

#### 4.6 PROCEDURE

For the in-person data collection, we used the paper questionnaires and requested 104 participants to use a pencil (~11%) and 231 participants (~24%) to use a pen to sketch UI elements. Similarly, we requested 246 participants (~25%) to digitally draw UI element sketches using a stylus with our cross-platform iOS and Android tablet application (Table 4.1).

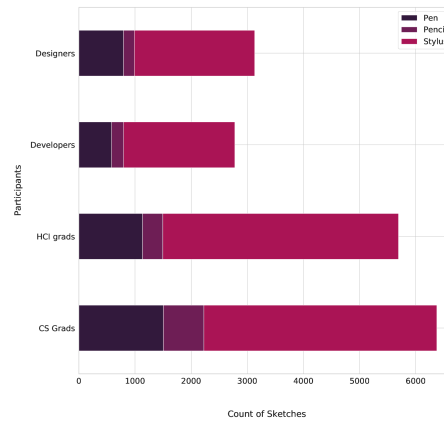
To collect data from a massive amount of participants in-person, we conducted eight data collection sessions with more than 50 participants in each session. We conducted these sessions in Germany, India, and Pakistan. Attendees were compensated for their participation. The study took approximately 15 minutes. However, participants were allowed to take more time, if need be.

For remote studies with 386 participants (~40%), we used our web application, which allows only stylus input. To ensure a diverse dataset of sketches within each UI element category, we restricted the number of sketches a participant could draw to one per UI element category. We collected data from participants belonging to Germany, Russia, China, Albania, India, Pakistan, United Kingdom, Canada, Iran and the U.S.A. The duration and compensation were the same as for in-person data collection sessions.

In totality, we collected 18,373 sketches for 19 UI elements from 967 participants. We used digital questionnaires for 632 participants and paper questionnaires for 335 participants (Table 4.1, Figure 4.3).

At the end of the data collection process, we had two different types of data: UI element sketches from the paper questionnaires and UI element

*Most of the data were collected by in-person data-collection sessions in universities, research institutes, conferences, and meetups. The rest were collected remotely by spreading digital questionnaires.*



**Figure 4.3:** Count of UI element sketches collected from different type of participants, split by the medium that they used to sketch

sketches from the digital questionnaires. The next step was to digitize the UI element sketches collected using paper questionnaires to merge them with sketches collected using digital questionnaires.

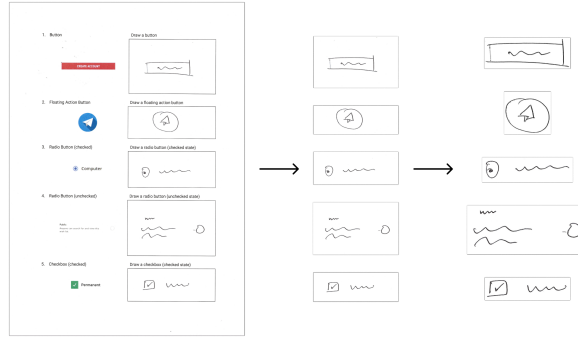
#### 4.7 DATA PROCESSING

To digitize the sketches collected using paper questionnaires, we scanned all the paper questionnaires with 600 DPI settings (4958x7008) and stored each sheet as a JPEG image. Next, we utilized a rectangular contour detection algorithm to extract the rectangular area containing UI element sketches from the paper questionnaire scans and discard unessential content (Figure 4.4). Afterwards, we manually labelled these UI element sketches and merged them with the unprocessed labelled sketches collected using the digital questionnaires.

*The data processing from paper questionnaires were automated using a pattern recognition algorithm that detects contours and crops them.*

We further processed this entire collection of sketches to obtain close-cropped UI element sketches. For this, we created a python script that takes the unprocessed labelled dataset as input and returns labelled close-cropped UI element sketches (Figure 4.4). This script applies Gaussian filtering with a large kernel (set to 10% of the image dimension) followed by an OpenCV contour detection algorithm to obtain close-cropped sketches. We later manually rechecked and cropped any sketches missed or incorrectly cropped by this algorithm. As a result, we obtained a processed labelled dataset with 18,373 UI element sketches.

As per the lessons learned from the pilot study, we expanded this dataset by adding sketches of images and labels. We obtained these sketches by cropping sketches of images and labels from other standalone and com-



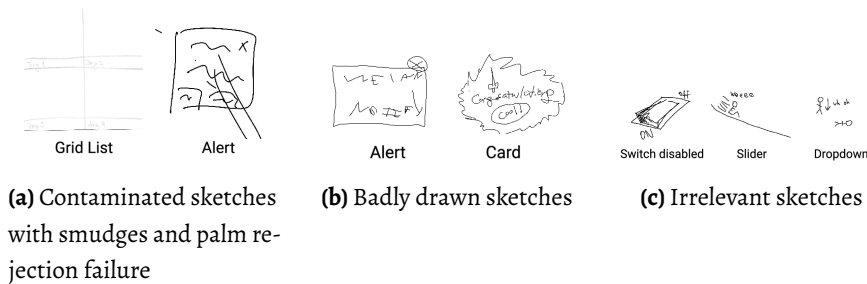
**Figure 4.4:** Process of extracting the close-cropped UI element sketches from the paper questionnaire

posite UI element sketches, e.g., button, text field, text area, grid list, card, and alert. By doing this step, we added 1,707 sketches of image and label to the dataset (Table 4.1). As a result, we expanded our dataset to contain sketches of all 21 UI elements.

#### 4.8 DATA VERIFICATION

As a next step, we manually cross-checked and cleaned this dataset. We discarded sketches that contained profanity or unrelated content (Figure 4.5c). Additionally, we rejected other few sketches based on image clarity and quality (Figure 4.5a,b). However, we did not remove UI element sketches merely because they were traced or not neatly drawn. As a result of data verification, in total, 2,101 UI element sketches (11.43%) were discarded.

*We manually cleaned the dataset by discarding contaminated, poorly drawn and irrelevant sketches.*



**Figure 4.5:** Examples of rejected sketches due to contamination, badly drawn or irrelevant content.

We truncated our dataset to contain an average of ~856 sketches (SD=13.73, Median=850) per category resulting in our final dataset of 17,979 UI element sketches. We sized the UI element categories uniformly to simplify

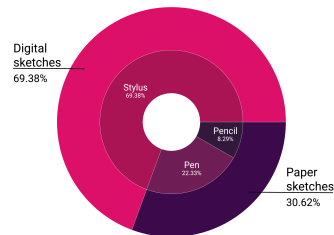
the training and testing process by avoiding the need to correct for bias towards the larger categories when training and validating a classifier.

#### 4.9 COLLECTED DATA

Finally, by following these data processing and verification procedures, we obtained the UISketch dataset. This dataset contains **17,979** close-cropped UI element sketches of **21 categories of UI elements** (Table 4.2) split into 69.38% of digital sketches and 30.62% paper sketches. Specific to the medium of sketching, it has 22.33% of pen sketches, 8.29% of pencil sketches and 69.38% of stylus sketches. Each UI element category contains on average ~856 sketches (SD=13.73, Median=850).

**Table 4.1:** Overview of data collection and verification process in terms of quantity of UI element sketches collected and extracted

Count	Pen	Pencil	Stylus	Total
Participants	231	104	632	967
Collected sketches (19 UI elements)	4,389	1,976	12,008	18,373
Images	194	72	581	847
Labels	193	79	588	860
Removed during verification	761	637	703	2,101
<b>Final dataset (21 UI elements)</b>	<b>4,015</b>	<b>1,490</b>	<b>12,474</b>	<b>17,979</b>



**Figure 4.6:** Distribution of digitally drawn UI element sketches using stylus vs. paper sketches drawn using pen and pencil in UISketch dataset

**Table 4.2:** Count of sketches collected for each UI element category

UI Element Category	Count
Button	881
Checkbox Checked	883
Checkbox Unchecked	872
Dropdown Menu	848
Floating Action Button	877
Image	847
Label	860
Radio Button Unchecked	867
Radio Button Checked	862
Slider	863
Switch Enabled	862
Switch Disabled	847
Text Area	853
Text Field	841
Alert	850
Card	844
Chip	850
Data Table	836
Grid List	840
Menu	849
Tooltip	847
<b>Total</b>	<b>17,979</b>

## 4.10 APPLICATIONS

This dataset is an initial exploration of how UI/UX designers sketch and recognize UI elements. Additionally, computational sketch recognition enables this dataset to have several interesting applications.

**SYNTHETIC DATASET** Recently many research projects have attempted to convert lo-fi sketches to code by using Deep Neural Networks (DNNs) (Section 2.3). Training such DNN models requires large scale datasets with ground truth sketches and their respective annotations. However, there is a lack of a large dataset to achieve good precision for this task. Synthetic data is a possible alternative to real data used by machine learning practitioners in situations where collecting real data is challenging either due to time, funds, or privacy limitations (Albuquerque et al., 2011). This dataset can be used to generate a large synthetic dataset to train such object detection networks.

**SKETCH-BASED IMAGE RETRIEVAL** This dataset can also be utilized in sketch-based image retrieval (SBIR) similar to SWIRE dataset by F. Huang et al. (2019). Current UI design libraries allow designers to search through their extensive database with textual keywords. With an SBIR system, the designers could query UI design screenshots in these large databases by quickly sketching their desired UI element categories. In addition to being an alternative to keyword-search, SBIR also allows matching the UI element category's style. This dataset enables training such systems by matching the UI element sketches with an appropriate UI design.

**SKETCH COMPLETION** Another application of this dataset can be sketch completion. In addition to the paper and digital sketches of UI elements, this dataset contains stroke information of the digital UI element sketches. AI trained with this dataset can assist in auto-completing UI element sketches as the trained neural network can classify what the designers are drawing while creating a stroke.

## 4.11 SUMMARY

In this chapter, we described the data processing and verification procedures we followed to obtain the UISketch dataset. This dataset is the

first large-scale dataset of 17,979 hand-drawn sketches of 21 UI element categories collected from 967 participants, including UI/UX designers, front-end developers, HCI, and CS grad students, from 10 different countries. This dataset aims to pave the way to further work on assisting UI designers during the prototyping process by creating and training new DNN models for sketch completion, sketch-based UI retrieval and converting lo-fi sketches to code.



---

## LOFI SKETCH DATASET

---

This chapter introduces the LoFi Sketch dataset, the first large-scale dataset of 4,527 hand-drawn annotated LoFi sketches. These sketches were collected from 361 participants, including UI/UX designers, front-end developers, HCI, and CS grad students, from 76 countries. This dataset contains LoFi sketches in both raster and vector format. In the following sections, we briefly explain the design decisions we made and the methodology we followed to collect and process this dataset. Excerpts from this chapter were published as our research paper, (Pandian, Shams, et al., [2022](#)) and as a thesis dissertation (Shams, [2021](#)) created under my supervision based on my research.

### 5.1 OBJECTIVE

In the previous chapter, we introduced the UISketch dataset, a large-scale collection of UI element sketches. Unlike UI element sketches, a LoFi sketch represents a complete UI screen depicted by designers during the initial prototyping phase. Therefore, in addition to the UISketch dataset, we aim to collect LoFi sketches to analyze how designers understand and communicate UI design. The primary objective of the LoFi sketch dataset is to obtain a large dataset of real-life hand-drawn annotated LoFi sketches from diverse participants using different input mediums such as pen, pencil, mouse, finger, and stylus. Also, we aim to obtain priors sketches made by designers during their prototyping process. Thus, these sketches reflect the natural LoFi prototyping process and help to validate the ecological validity and benchmark AI models. Further, this dataset enables UI element detection, UI sketch auto-completion and LoFi sketch-based screenshot retrieval research.

## 5.2 DESIGN DECISIONS

*In addition to paper and digital questionnaires, we also collected LoFi sketches from participants who provided their prior LoFi prototypes made while creating their application GUIs.*

*We allowed participants to sketch using pen, pencil, stylus, mouse, and touch.*

Similar to this thesis, we scoped this dataset for LoFi sketches of smart-phone applications. Based on the lessons learned from our prior research on collecting UI element sketches (Chapter 4) and by the data collection practice used by F. Huang et al. (2019), we decided to provide a sample UI screenshot to the participants from the RICO dataset and ask them to sketch it as a LoFi prototype. Therefore, we collected and annotated 100 UI screenshots from the RICO dataset. However, we aimed to collect a diverse dataset representing scenarios where LoFi sketches were made from UI design inspirations and real-life scenarios where LoFi sketches were made. Therefore, to obtain ecologically correct samples, we also collected LoFi sketches of real LoFi prototypes that our participants made in addition to the data collection.

Regarding the medium of sketching, we collected the digital LoFi sketches and paper LoFi sketches. In digital sketches, we allowed participants to sketch the prototype using either stylus, touch, or mouse as input, and in paper-based sketches, we allowed participants to use pen and pencil. With these design decision, we started the data collection by creating paper and digital questionnaires.

## 5.3 DATA COLLECTION QUESTIONNAIRES

We repurposed the paper and digital questionnaire from the UI element data collection (Section 4.3) to collect the LoFi sketch dataset.

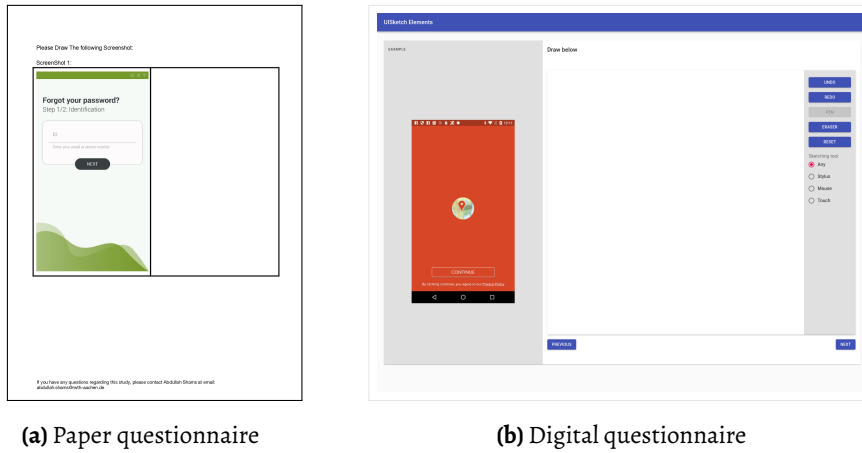
We created the paper questionnaire<sup>1</sup> such that each question has an example of a UI screenshot on the left and a rectangular area to sketch the UI element on the right-hand side (Figure 5.1a). These example images were randomly selected from the 100 UI screenshots we picked from the RICO dataset and intended to assist the participants by providing a UI design inspiration.

*Similar to the UISketch dataset, we provided UI screenshot samples and asked participants to use them as inspiration to create LoFi sketches.*

We modified the web application, earlier used to collect the UISketch dataset, to create the LoFi sketch digital questionnaire<sup>2</sup>. Unlike its predecessor, this questionnaire allows mouse, touch and stylus input to sketch LoFi prototypes (Figure 5.1b). The web application starts by requesting participants to provide informed consent and provides an informative video to understand the purpose of this data collection. Once the participant

<sup>1</sup> [https://blackbox-toolkit.com/f/lofisketch/paper\\_questionnaire.pdf](https://blackbox-toolkit.com/f/lofisketch/paper_questionnaire.pdf)

<sup>2</sup> <https://uisketch-lofi.web.app>



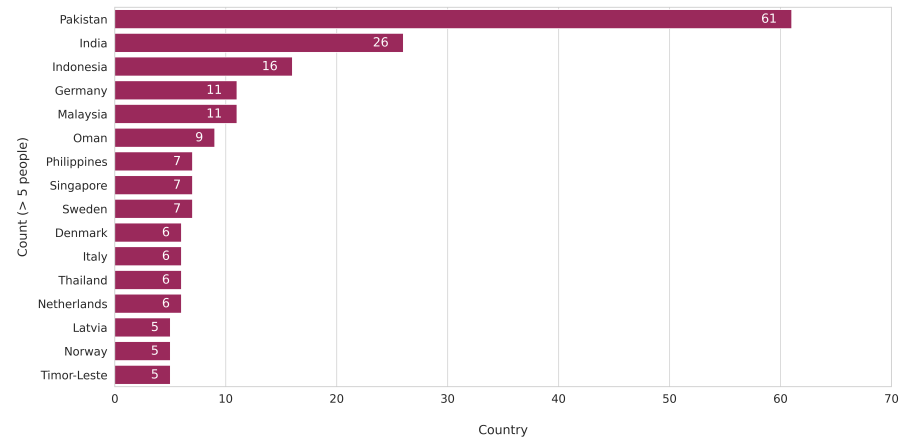
**Figure 5.1:** Paper and digital questionnaire used for collecting LoFi sketch dataset. Both questionnaires have an example of a inspiration UI screenshot on the left and a rectangular area to sketch the LoFi on the right-hand side.

agrees to the consent form, similar to the earlier versions, this web application creates a secure key for authentication and stores it in their browser's internal database. This step restricts the participants from taking part in the study multiple times. It also helps restore the web application from the point they left off in case of internet disconnection.

#### 5.4 PARTICIPANTS

With the paper and digital questionnaires, we collected LoFi sketches from 361 participants: 30 UI/UX designers with 1-8 years ( $M=2.83$ ,  $SD=1.54$ ) of prior prototyping experience, five front-end developers with 2-5 years ( $M=2.8$ ,  $SD=1.3$ ) of prior UI development experience, 44 programmers with 1-44 years ( $M=4.39$ ,  $STD=6.70$ ) of GUI and general programming experience, 251 HCI graduate students with up to one year of UI prototyping experience, and 31 Computer Science graduate students who had attended the UI design and development related courses. We did not restrict data collection based on the age or gender of the participants. We collected data from participants belonging to 76 countries: Pakistan, India, Indonesia, Germany, Malaysia, Oman, Sweden, Singapore, Philippines, etc. (Figure 5.2). We used purposive and snowball sampling to recruit these participants. We contacted several UI/UX professionals and developer through social media and online platforms such as Reddit, Facebook, LinkedIn, and Twitter for gathering such a massive amount of participants. We also reached out to several students by contacting Human-Computer Interac-

*We collected the LoFi sketches using paper and digital questionnaires from 361 participants, including UI/UX designers, front-end developers, programmers, HCI, and CS grad students, from 76 different countries.*



**Figure 5.2:** Count of participants ( $\geq 5$  people) per country that contributed to LoFi sketch dataset

tion (HCI) and Computer Science (CS) Professors from various universities. Additionally, we approached companies and institutes working in the HCI domain.

## 5.5 PROCEDURE

As we aimed to collect a diverse dataset representing design inspiration based sketches and real-life LoFi sketches, we conducted the data collection procedure in two different ways: collection sketches through questionnaires, acquiring premade LoFi sketches from participants. Table 5.1 summarizes the count of LoFi sketches collected by various procedures. In this section, we discuss both the procedures successively.

### 5.5.1 Questionnaire-based

The nationwide lockdown in the years 2020 and 2021 due to COVID-19 limited our data collection process; therefore, we formulated our data collection procedure accordingly. As in-person meetings are restricted, we concentrated on collecting data through digital questionnaires.

The initial steps data collection procedure for both paper and digital questionnaires were similar. We explained to the participants about the reason and benefits of the study and asked them to provide us with their demographics (optional) and consent (mandatory) to participate in the study. We then explained how the questionnaire and how to use it. We requested the participants not to trace the UI screenshot example, instead

*We collected most of the data (86%) from paper and digital questionnaires.*

**Table 5.1:** Split of participants and count of collected sketches based on data collection source

	Participants	Count of collected LoFi Sketches
Digital Questionnaire	343	8,860
Paper Questionnaire	18	214
Other sources	-	1,201
<b>Total</b>	<b>361</b>	<b>10,275</b>

use it as an inspiration to model their LoFi sketch based on it. We did not restrict the number of LoFi sketches the user could draw, and then we terminated the study once the participant decides to stop the study. We informed them that they could also resume the study at any later time. We also provided the participants the flexibility to skip a UI design sample if they were not comfortable with sketching it. Although this approach provided us with unequal distribution of LoFi sketches drawn by participants, this allowed the participants from being overwhelmed by the task.

Using this procedure, we collected data from 18 participants (~5%) using paper questionnaires and 343 participants (~95%) using digital questionnaires. In total, we collected 214 paper-based LoFi sketches and 8,860 digital LoFi sketches using the questionnaires.

#### 5.5.2 Other sources

We contacted several universities, research institutes, and companies to acquire LoFi sketches. As LoFi sketching is a conventional process followed before building real-life GUI applications, we requested them to forward LoFi sketches made by UI/UX designers or HCI grad students for their previous projects. Due to the privacy policy, most companies and research institutes refused to provide their LoFi sketches. Few companies and research institutes provided the sketches but without the demographic details. We also acquired few LoFi sketches students drew for their course capstone projects.

In total, we acquired 695 paper-based LoFi sketches by this process. However, we could map this information neither to participants nor their demographics. Therefore, this information is not well-represented in the participant information.

**Table 5.2:** Overview of sketches removed during LoFi sketch data verification process

Sketches	Count of LoFi sketches
Collected	10,275
With little to no details	3,027
With rating < 5	2,721
<b>LoFi sketch dataset</b>	<b>4,527</b>

## 5.6 DATA VERIFICATION

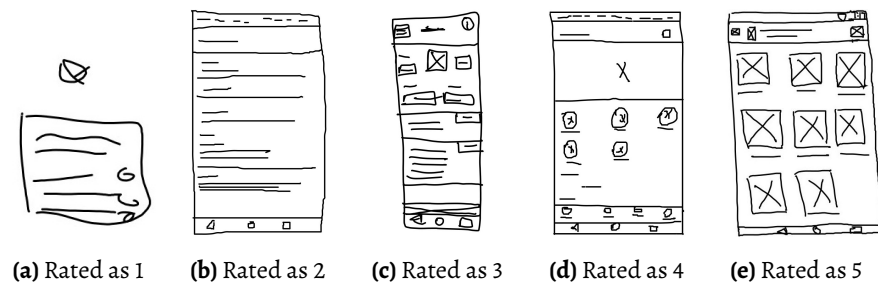
From our data collection process, we collected 10,275 LoFi sketches from various sources. As a next step, we manually merged all the collected LoFi sketches, reviewed, and cleaned them.

In digital questionnaires, many participants decided to skip sketching a sample by leaving the sketching area empty or leaving the sketch unfinished. Therefore, we started the data verification process by removing the sketches with little to no detail. In this step, we identified and discarded 3,027 LoFi sketches (29.46%).

*We discarded sketches with little to no detail, rated the LoFi sketches, and chose only the best-rated sketches.*

Also, as the details present in a LoFi sketch varies due to the input medium (pen, pencil, mouse, touch, and stylus), we added rating to the sketches on a scale of 1 to 5, with one being the worst and five being best. We then discarded all the sketches with a rating less than 5. Figure 5.3 shows a sample LoFi sketch with each rating in our scale. By this process, we removed 2,721 LoFi sketches (26.48%). We performed this step to improve the quality of the sketches in the final LoFi sketch dataset.

Through the data verification steps described above, we removed 5,748 LoFi sketches (55.94%) from the 10,275 LoFi sketches collected from various sources and finally accepted 4,527 sketches in the LoFi sketch dataset. Table

**Figure 5.3:** Samples of LoFi sketches rated in a scale of 1 to 5.

5.2 summarizes the count of LoFi sketches collected and removed by the data verification process.

## 5.7 DATA ANNOTATION

Our goal for creating the LoFi sketch dataset is to provide a large-scale annotated dataset for training DNN models. Therefore, after the data verification process, we created annotations for each LoFi sketch in the dataset.

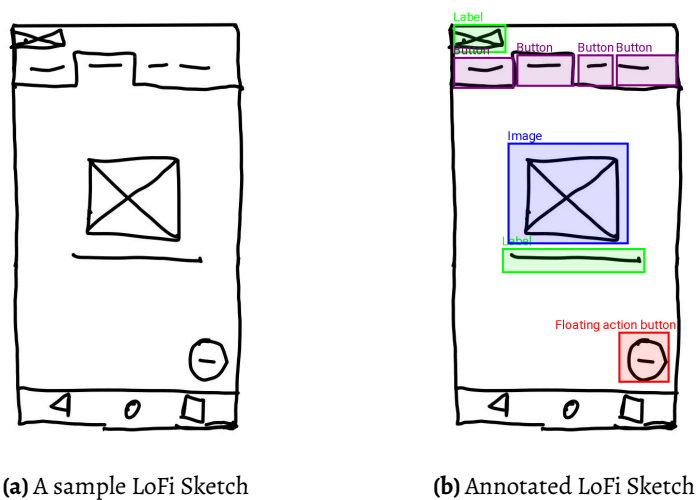
A LoFi sketch is made of various UI elements present in different locations and has different dimensions. Through this dataset, we aimed to capture the information of the constituent UI elements in a LoFi sketch along with their location and dimension in that sketch. To annotate these sketches, we used the Supervisely<sup>3</sup> platform. We loaded the 4,527 LoFi sketches in Supervisely and annotated them into the 21 UI element categories specified in the UISketch dataset (Table 4.2). Figure 5.4 shows a sample LoFi sketch with its annotations.

*Through annotation, we identified 41,560 constituent UI elements in the 4,527 LoFi sketches.*

## 5.8 COLLECTED DATA

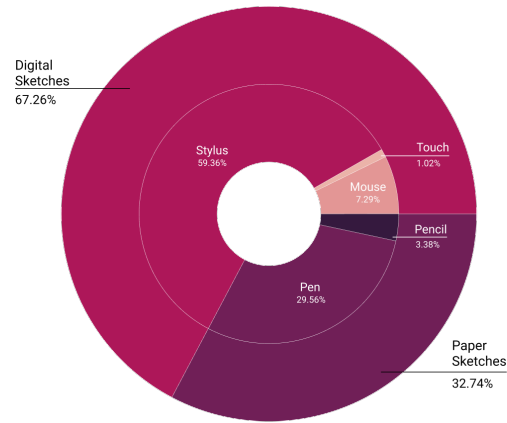
With the annotation files, we created the LoFi sketch dataset. This dataset contains 4,527 LoFi sketches annotated with 41,560 constituent UI elements classified into 21 categories. On average, each LoFi sketch contains

<sup>3</sup> <https://supervise.ly/>



**Figure 5.4:** Sample LoFi Sketch with its annotations

9.18 UI elements ( $SD=5.72$ ,  $Median=8$ ). Most of the LoFi sketches were collected from paper and digital questionnaires using pen, pencil, mouse, touch, and stylus as the input medium. Figure 5.5 visualizes the distribution of LoFi sketches collected using the questionnaires based on the medium.



**Figure 5.5:** Distribution of digitally drawn LoFi sketches using stylus, mouse, and touch vs. paper sketches drawn using pen and pencil in LoFi sketch dataset

## 5.9 BENEFITS & APPLICATIONS

LoFi sketch is the first large-scale annotated LoFi sketches dataset. This dataset opens up several possible research areas.

**UI ELEMENT DETECTION** As seen in the literature review, many research projects have attempted to convert LoFi sketches to code using DNN models (Section 2.3). Although this dataset is not enough as stand-alone to train such DNN models, it can be combined with synthetic datasets (discussed in the previous chapter) to create generalizable UI element detection networks.

**SKETCH COMPLETION** Y. Li et al. (2020) explored a novel research area of auto-completing UI wireframes. This dataset contains rich information about UI element categories and their layout in a LoFi sketch; therefore, it can be utilized for training auto-completion DNNs which supports designers by autocompleting their strokes.



## 5.10 SUMMARY

In this chapter, we introduced the LoFi sketch dataset, the first large-scale dataset of annotated LoFi sketches. This dataset contains 4,527 LoFi sketches annotated with 41,560 constituent UI elements. We collected these sketches from 361 participants from 76 countries. These LoFi sketches were collected from various sources and were sketched in various mediums. Most of the LoFi sketches were collected from paper and digital questionnaires using pen, pencil, mouse, touch, and stylus as the input medium. The rest of the LoFi sketches were collected from participants who provided their prior LoFi sketches made while creating their GUIs. This dataset enables further research on supporting UI designers during the prototyping process by creating and training DNN models for converting lo-fi sketches to code and auto-completing LoFi sketches.



---

## SYN & SYNZ DATASETS

---

This chapter discusses two datasets, Syn<sup>1</sup> and SynZ<sup>2</sup>, which provide large scale synthetic annotated LoFi sketch datasets for training DNN models.

The excerpts from this chapter were published as a part of our research papers “Syn” (Pandian, Suleri, and Jarke, 2020) and “SynZ Dataset” (Pandian, Suleri, and Jarke, 2021a). We describe both Syn and SynZ successively in the upcoming sections, describing its synthetic data generation methodology, advantages, and its statistical description.

### 6.1 OBJECTIVE

The primary objective behind generating both Syn and SynZ datasets is to provide a large-scale dataset for training data-demanding AI models. In general, synthetic data is a possible alternative to real data used by machine learning practitioners in situations where collecting real data is challenging either due to time, funds, or privacy limitations (Albuquerque et al., 2011). As obtaining LoFi sketches in the hundred-thousands is a laborious and expensive task, the Syn and SynZ dataset provides a base for training AI models which can be fine-tuned further by a relatively smaller real-life LoFi dataset. Therefore, this dataset opens up various avenues in pretraining AI models such as UI element detection and UI layout refinement.

### 6.2 SYN

Syn<sup>1</sup> is a synthetic LoFi sketch dataset containing 125,000 LoFi sketches. These LoFi sketches were synthetically generated by randomly allocating the UI element sketches from the UISketch dataset (Chapter 4) with 17,979

---

<sup>1</sup> <https://www.kaggle.com/vinothpandian/syn-dataset>

<sup>2</sup> <https://www.kaggle.com/vinothpandian/synz-dataset>

UI element sketches of 21 categories of UI elements collected from 967 participants. In the upcoming subsections, we describe the algorithm we used to generate Syn and the advantages of Syn.

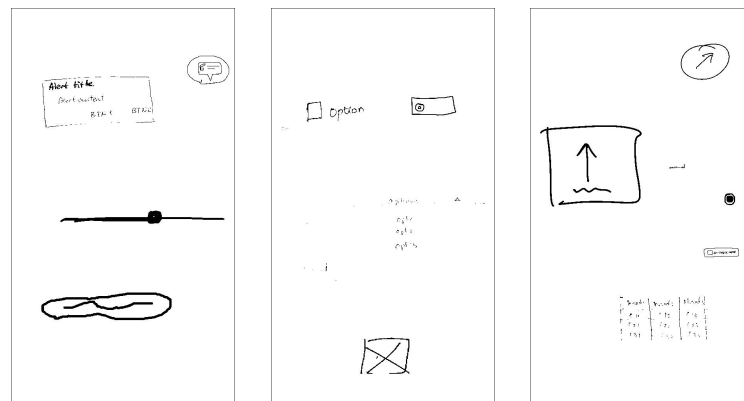
### 6.2.1 Data generation process

*Syn was generated by random allocation of position and dimension of UI element sketches in a blank image.*

To generate Syn, we created a script<sup>3</sup> that randomly samples 1 to 15 UI element sketches from the UISketch dataset. Then, it calculates the total area of these randomly sampled UI element sketches and scales it by a factor ranging from 2 to 4. It then creates a blank image with this scaled area as image dimensions. The script then attempts to assign a non-overlapping position to each randomly sampled UI element sketch in this blank image by back-propagation. Once it successfully identifies the location for each UI element sketch, it stitches this UI element sketch in the blank image. As a result, we get a synthetically generated LoFi sketch (Figure 6.1).

In addition to generating the synthetic images, this script also creates the respective annotation files in two different formats: CSV and COCO dataset format. These annotation files contain all the constituent UI elements along with their respective location and dimension for a given synthetic LoFi sketch. We provide the annotations in multiple formats to enable this module to be coupled with any open-source object detection API. Using this procedure, we synthetically generated **125,000** LoFi sketches along with their annotation files and provided it as an open-source dataset, **Syn**.

<sup>3</sup> <https://blackbox-toolkit.com/f/uisketch/datagen.py>



**Figure 6.1:** Sample generated synthetic LoFi sketches from the Syn dataset

### 6.2.2 Advantages

There are three major benefits of using this approach. They are listed below.

- We can synthetically generate a large-scale training dataset containing more than 125,000 annotated LoFi sketches by permuting the 17,797 UI element sketches we collected via paper and digital questionnaires. This technique reduces the time and effort in annotating LoFi sketches.
- Randomly scaling UI elements before assigning them a position on the blank image aligns with the data augmentation strategy used in training object detection models for boosting the model performance as described by W. Liu et al. (2016). This strategy helps in pre-training and later fine-tuning a model with a smaller real-life LoFi sketch dataset.
- Similarly, randomly positioning UI elements in an annotated image helps in fine-tuning multi-box detection methods to detect objects at any given location as proposed by W. Liu et al. (2016). Thus enabling the DNN model to learn the distribution of UI elements without over-fitting.

### 6.2.3 Generated data in Syn

Syn is the first large-scale synthetic dataset of LoFi sketches containing 125,000 LoFi sketches with 21 categories of UI elements. Even though the random allocation of UI element sketches is advantageous in the initial training of DNN models, one major shortcoming of Syn is that it randomizes the size, location, and distribution of each UI element bounding box in every generated UI sketch. This randomization does not statistically correspond to real-life UI sketches drawn by UI/UX designers. This lead to our need to generate SynZ, a large-scale dataset that is statistically similar to real-life UI sketches. We discuss SynZ dataset generation and its advantages in the next section.

*Syn contains 125,000 LoFi sketches with 21 categories of UI elements. However, Syn does not statistically correspond to real-life UI sketches drawn by UI/UX designers.*

### 6.3 SYNZ

*SynZ was created to be coupled with Syn and overcome its shortcomings.*

This section introduces the SynZ dataset<sup>2</sup>, which contains 175,377 synthetically generated UI sketches statistically similar to real-life UI screens. To generate SynZ, we analyzed, enhanced, and extracted annotations from the RICO dataset and used 17,979 hand-drawn UI element sketches from the UISketch dataset. We explain this process in detail below and visualize it in Figure 6.2

#### 6.3.1 Enhancing RICO annotations

We started by analyzing the existing annotations from the RICO dataset. Based on our analysis, we followed the steps discussed below to enhance the RICO dataset annotations.

##### 6.3.1.1 Removing Outliers

These annotations are extracted from the corresponding view hierarchies. Consequently, the annotations had certain anomalies such as unusually large (e.g., Switch on/off that fits the entire UI screen), small (area approx equal to zero), and erroneous (size less than zero) bounding boxes. Therefore, we calculated each bounding box area and removed the outlier annotations using the Inter-Quartile Range (IQR) method.

##### 6.3.1.2 Mapping UI Element Categories

We observed that a few UI element categories specified in RICO annotations are specific to front-end code and not relevant to UI sketches, such as Map View, Pager Indicator, and Advertisement. Also, a few composite UI components such as Toolbar and Button bar can be represented as a collection of standalone UI elements such as Buttons and Images. Moreover, these UI element categories do not map to the UI element categories from the UISketch dataset. Therefore, we mapped the annotations from RICO to the UISketch dataset to obtain a unified collection of annotations of 21 UI element categories.



**Figure 6.2:** Flowchart visualizing different steps involved in modifying RICO annotations to SynZ dataset

### 6.3.1.3 Adding UI Element States

The RICO dataset annotations do not contain any information regarding the state of UI elements, such as Checkbox checked/unchecked. Therefore, we equally split the annotations of each stateful UI element category (e.g., Checkbox, Switch, Radio Button) as enabled or disabled.

### 6.3.1.4 Adding New UI Elements

Also, RICO does not distinguish text field as dropdown, or text area, so as text areas are by condition bigger than text fields, we considered all text fields above 75 percentile area as text areas. Then we equally split the remaining text field annotations into dropdowns and text fields. Similarly, we split large labels (area greater than 75 percentiles) as chips.

We also observed that RICO did not contain a few UI element categories such as FAB, Tooltip, Grid List, and Data Table present in the UISketch dataset. Tooltips are generally placed above/below UI elements. Therefore we identified all possible positions of a Tooltip algorithmically and added the bounding boxes. The Grid List and Data Table are full-width UI elements that occupy almost 90% of the UI screen. Therefore, we identified all possible screens with the appropriate free area and allocated them as Grid Lists and Data Tables.

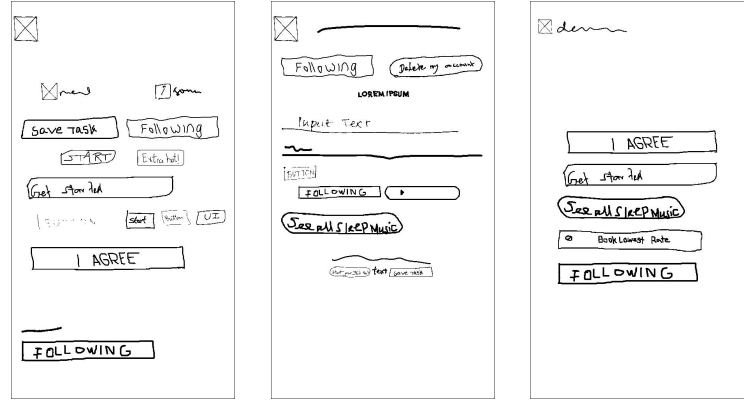
With these steps, we created SynZ annotations by enhancing the RICO dataset annotations. These SynZ annotations contain 58,459 UI screen annotations with 547,933 UI element bounding boxes for 21 UI element categories. On average, each UI screen has 9.35 UI element bounding boxes (SD=8.05).

*We extracted and enhanced annotations from the RICO dataset to generate SynZ annotations.*

### 6.3.2 Data generation process

To synthetically generate UI sketches for the SynZ dataset, we used the latest UISketch dataset, which contains 17,979 sketches of 21 UI element categories drawn by 967 experienced designers, developers, and grad students. For each UI sketch in Synz, we created a blank image with dimensions 1440x2560, similar to the standard size of UI screenshot from RICO. For each UI element annotation present in this UI screenshot, we randomly sampled a UI element sketch of the given UI element category (without replacement) from the UISketch dataset. We then rescaled this UI element sketch with the aspect ratio to fit within the corresponding bounding box and stitched it in the blank image. Then we resized this image keeping the

*SynZ dataset was generated by replicating the UI element distribution in the RICO dataset annotations.*



**Figure 6.3:** Sample generated synthetic LoFi sketches from the SynZ dataset

aspect ratio to 640x360. Figure 6.3 shows few samples of synthetic LoFi sketches from the SynZ dataset.

With this process, we used the UISketch dataset to generate the SynZ dataset of UI Sketches along with their annotations. We reused the same UI screenshot annotations from the RICO dataset three times to generate a substantial amount of UI sketches. As each iteration uses a different random state, the samples returned are almost always different.

This procedure enabled us to generate 175,377 UI sketches and their annotations in COCO dataset format and CSV format. For transparency and replicability of this research, we have open-sourced the codebase<sup>4</sup>.

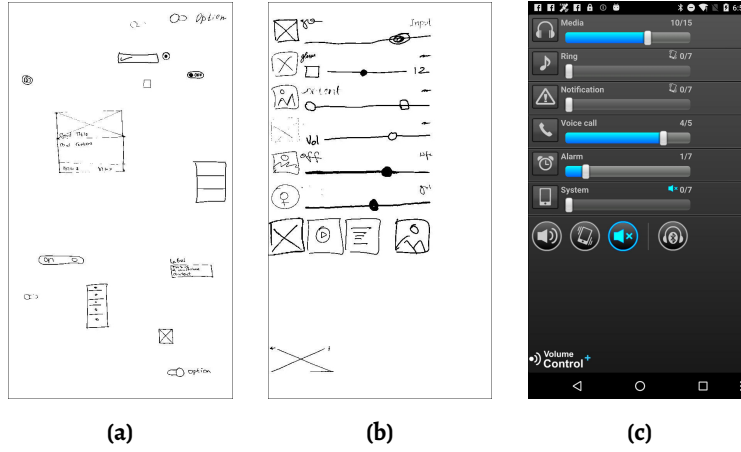
### 6.3.3 Advantages

There are four major benefits of generating the SynZ dataset. They are listed below.

- Similar to Syn, SynZ dataset generation allows us to generate more than 175k synthetic LoFi sketches from the UISketch dataset.
- SynZ coupled with Syn provides a robust base for training DNN models before fine-tuning with real-life LoFi sketch dataset.
- As the SynZ annotations are extracted by enhancing the RICO annotations; they follow a similar UI elements distribution as the RICO dataset. Therefore, the UI sketches synthetically generated by SynZ are statistically similar to real-life UI screens collected in the RICO dataset. This dataset is in line with the two fundamental requirements of synthetic data prescribed by Patki et al. (2016): 1) it must

<sup>4</sup> <https://github.com/vinothpandian/synz>





**Figure 6.4:** Sample representative UI images from (a) Syn, (b) SynZ, (c) RICO

statistically approximate the real data, 2) must be structurally similar to the real data.

- The enhanced RICO annotations obtained from the process of generating SynZ is not just limited to LoFi sketches. Therefore, the annotations can be used to understand the UI element distribution in real-life smartphone UI design.

#### 6.3.4 Generated data in SynZ

In this section, we described SynZ, a large-scale open-access dataset of 175,377 UI sketches synthetically generated using 17,979 UI element sketches from the UISketch dataset drawn by 967 participants. Initially, we analyzed 72K UI screens from the RICO dataset and enhanced its annotations by removing outliers and adding new UI element categories and states. Later, we utilized these enhanced annotations from the RICO dataset to generate the SynZ dataset. Therefore, the UI sketches in SynZ are statistically similar to the real-life UI screens.

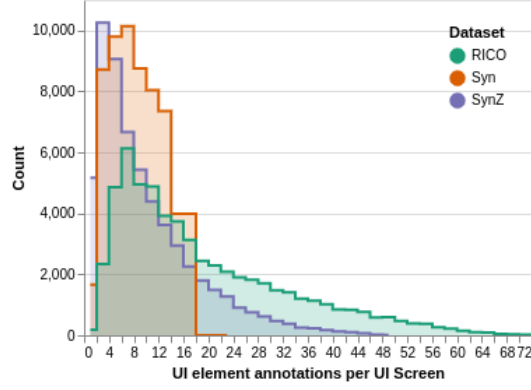
### 6.4 COMPARISON OF SYN AND SYNZ DATASETS

Although Syn and SynZ are both synthetically generated and serve the same purpose, the difference between them lies in their statistical similarity to real-life LoFi sketches and the advantages they provide. Syn dataset contains 125,000 synthetic LoFi sketches generated algorithmically by random allocation of UI elements and their position. In contrast, the SynZ dataset contains 175,000 synthetic LoFi sketches generated by statistical

*In contrast to Syn, the SynZ dataset is statistically similar to the real-life UI screens. However, they can be coupled together to train DNNs to utilize their potential advantages.*

**Table 6.1:** Comparison of of Syn and SynZ datasets

Dataset	UI sketches	UI elements in a UI sketch	UI element categories	Bounding box locations
Syn	125,000	1-22	21	Random allocation
SynZ	175,377	1-49	21	Similar to real-life UI screens from RICO

**Figure 6.5:** Comparison of UI element distribution using in Syn vs SynZ represented by a histogram plot

analysis of the RICO dataset’s UI screenshots. As the SynZ annotations are extracted by enhancing the RICO annotations, they follow a similar UI elements distribution as the RICO dataset (Table 6.1, Figure 6.5 and 6.4). Therefore, the UI sketches synthetically generated by SynZ are statistically similar to real-life UI screens collected in the RICO dataset.

## 6.5 BENEFITS & APPLICATIONS

Synthetic data is a possible alternative to real data used by machine learning practitioners in situations where collecting real data is challenging either due to time, funds, or privacy limitations (Albuquerque et al., 2011). In total, both Syn and SynZ combined contain around 300,000 synthetic LoFi sketches. In addition to the benefits listed in this chapter, this large-scale dataset provides a base for training various DNN models. We have listed a few applications below.

**UI ELEMENT DETECTION** Recent research projects, in the automatic transformation of LoFi sketches to code, utilize DNN based object detection models to achieve their goal. Object detection models require

large-scale dataset to be trained effectively. Therefore, this dataset can be used to pre-train such object detection networks before fine-tuning with real-life LoFi sketches.

**UI LAYOUT REFINEMENT** As a byproduct of generating SynZ, we refined the UI element annotations from the RICO dataset and created 58,459 annotated UI screenshots. These annotated UI screenshots contain information about the position and dimension of 21 categories of UI elements. This position and dimension data can be utilized for pre-training UI layout refinement DNNs which can be further fine-tuned with real-life UI wireframe semantic annotations.

## 6.6 SUMMARY

This chapter described two large-scale synthetic LoFi sketch datasets that we contribute to this thesis, Syn and SynZ. Both Syn and SynZ datasets were generated using 17,979 UI element sketches from the UISketch dataset described in the previous chapter. Syn dataset contains 125,000 synthetic LoFi sketches generated by random allocation of UI elements in a LoFi sketch. Whereas, SynZ dataset contains 175,377 synthetic LoFi sketches generated by understanding the UI element distribution in UI screenshots from the RICO dataset. In contrast to Syn, the SynZ dataset represents the distribution of UI elements in a smartphone UI design better due to the underlying data generation strategy. Despite the difference, both Syn and SynZ datasets are vital for training a DNN model as they have their advantages and disadvantages.



---

## WIRED DATASET

---

This chapter introduces the final dataset contribution of this thesis, Wired dataset: a collection of semantic annotations of UI screenshots. This dataset contains 2,751 UI screenshots from the RICO dataset, categorized into 5 UI design patterns, and annotated with 100 semantic categories of UI elements. Unlike the previous three datasets, this dataset concerns with semantic layout information used to generate UI wireframes instead of LoFi sketches. Although LoFi sketches and UI wireframes layout the contents of UI design and their properties similarly, UI wireframes are more structured and refined than sketches. In contrast to LoFi sketches, wireframes use consistent design to denote a UI element, and their positions are aligned to the grid (Brown, 2011).

Semantic annotation of UI screenshots differs from the other bounding-box annotations as it contains specific information to semantically understand the context and nature of UI elements present in a wireframe. For example, a minimal template login screen contains a text field to get a username and another text field for password and finally, a button to submit the login information. An Android view hierarchy or basic annotation of this login page would contain the two UI element category (two text fields and one button), their location, and dimension. However, in semantic annotation of the same UI screenshot will the three UI element categories (a username field, a password field, and a login button), their group id if they are grouped, their location, and dimension. This detailed semantic information allows us to capture the intricacies and provides an excellent base for training powerful DNN models.

In the upcoming sections, we describe our rationale in scoping and categorizing the UI screenshots followed by the methodology we followed to create the Wired dataset. Excerpts from this chapter were published as a thesis dissertation (Gajjar, 2020) and our research paper title “Akin” (Gajjar et al., 2021) created under my supervision on the basis of my research.

*The Wired dataset contains 2,751 semantically annotated UI screenshots categorized into five UI design patterns.*

*Semantic annotations is an extension of basic UI annotations that contains specific information to semantically understand the context and nature of UI elements present in a wireframe.*

## 7.1 OBJECTIVE & DESIGN DECISIONS

RICO dataset collected by Deka et al. (2017) contains more than 72,000 Android smartphone screenshots and their respective Android view hierarchies. This dataset is categorized into 27 app categories. To understand the semantic information in the UI screenshots in RICO, T. F. Liu et al. (2018) and later Leiva et al. (2020) categorized the constituent UI elements in RICO dataset into 25 categories. The major shortcoming of the RICO dataset is that the annotations are automated, and therefore it contains various inconsistencies. As a solution, Leiva et al. (2020) recently revised a subset of UI screenshots from the RICO dataset, annotated these 1,460 UI screenshots manually, and classified them into 20 design topics to create the Enrico dataset.

The primary goal of the Wired dataset is to overcome these limitations of the RICO dataset. Through this dataset, we aim to provide manually refined, verified, classified and annotated UI screenshots obtained from the RICO dataset. Further, this dataset explores the context and nature of constituent UI element categories present in a UI screenshot. Therefore, this dataset can be reliably used for fine-tuning and benchmarking AI models in different domains such as UI wireframe generation and UI layout refinement.

*The Wired dataset is scoped and categorized to contain UI screenshots from e-commerce applications into five common UI design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page.*

*UI design patterns are used as lingua franca to discuss UI design concepts.*

In this thesis, we further this research domain by enhancing the annotations into semantic annotations. Additionally, we categorize these UI screenshots into UI design patterns and annotate them thoroughly by identifying 100 different semantic UI element categories. As there are more than 72k UI screenshots in RICO and 27 different app categories, which could be categorized into innumerable UI design pattern categories, we scoped this dataset into one app category, e-commerce applications. We further scoped and categorized the UI screenshots from e-commerce applications into five common UI design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page.

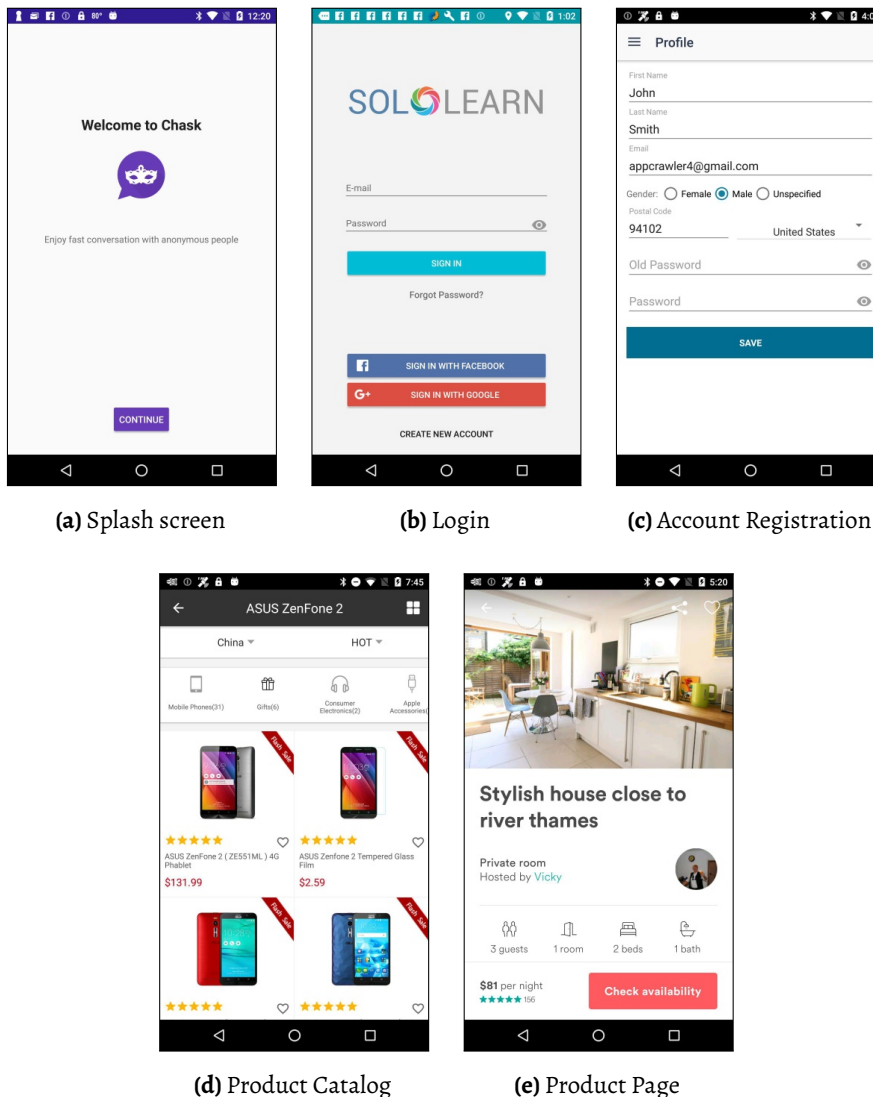
Tidwell et al. (2020) defines a UI Design Pattern as an entity that describes a reoccurring problem in UIs and proposes a solution to that problem. Thus Erickson (2000) claims that a UI design pattern acts as a common terminology (*lingua franca*) to refer to a specific UI design concept. The author further asserts that assigning UI design patterns as nomenclature for UI designs enables interdisciplinary design teams to communicate consistently and efficiently. Due to its benefits in communicating UI designs, Borchers (2002) and Seffah (2003) consider UI design patterns helpful in

sharing and discussing UI design and HCI knowledge to students. Therefore, categorizing UI screenshots into UI design patterns would benefit the HCI community and enable DNN models to be trained using supervised learning techniques.

In the next section, we discuss the strategy we used to select and categorize UI screenshots for the Wired dataset.

## 7.2 DATA CLASSIFICATION

As mentioned earlier, the RICO dataset contains over 72k UI screenshots; therefore, manually categorizing the UI screenshots is a laborious task.



**Figure 7.1:** A sample image of each UI design pattern from the dataset

**Table 7.1:** Count of UI screenshots per UI design pattern and UI element annotations per UI screenshot in the dataset

UI Design Pattern Category	Count of UI Screenshots	Count of UI element annotations
Splash screen	1,074	7,119
Login	798	9,359
Account Registration	498	6,226
Product Catalog	275	10,301
Product Page	106	2,178
<b>Total</b>	<b>2,751</b>	<b>35,183</b>

Therefore, we employed a semi-supervised DNN model to classify the UI screenshots and then manually verified the classifications.

*We used the VGG-16 model trained using semi-supervised learning technique to categorize UI screenshots into UI design patterns and then manually refined them.*

To train a DNN with a semi-supervised learning technique, we manually labelled 10,000 UI screenshots of the RICO dataset into six categories: Splash Screen, Login, Account Registration, Product Catalog, Product Page, and Others. This classification revealed that the entire RICO dataset was highly skewed towards the others category, with over 80% of them in that category. Splash screen and login screen categories had the second-largest share, with each around 7% of the screenshots. To counteract this class imbalance, we assigned a weight inversely proportional to its size in the dataset for each category. This weight is used as a multiplying factor for updating the gradients while training the semi-supervised DNN model.

After few empirical experiments with different classification models, we chose the VGG-16 model, introduced by Simonyan et al. (2015a), without pretrained weights for classifying the UI screenshots into 6 UI design patterns. This model was trained using the Adam optimizer with an initial learning rate of  $1 \times 10^{-4}$  and reducing it on a plateau by a factor of 0.2 and patience of 30 for 1000 epochs. Employing iterative label propagation and manual correction, we fine-tuned this model to classify a UI screenshot into the six chosen UI design pattern categories with an accuracy of 97%.

With this semi-supervised VGG-16 and manual verification, we selected and classified 2,751 UI screenshots into the five chosen UI design patterns. Figure 7.1 shows a sample image for each category and Table 7.1 shows the number of UI screenshots in each category. In the upcoming section, we discuss the annotation process of this dataset with 100 categories of UI elements.



### 7.3 DATA ANNOTATION

After categorizing the UI screenshots into five UI design patterns, we annotated the dataset to obtain semantic information of UI elements in each UI screenshot.

As discussed in section 2.2.1, each UI screenshot in the RICO dataset has a corresponding Android view hierarchy file that contains the UI elements in a hierarchy of UI element groups. The root level of this tree-like Android view hierarchy contains the Android-specific layout elements (such as relative layout, linear layout), whereas the leaf nodes contain UI elements (such as Button, Text field) visible on the screen. However, numerous RICO view hierarchy annotations are incorrect or partially labelled. Therefore, we manually annotated and verified the UI elements in each of the 2,751 UI screenshots we selected from the previous section. We used Labelme, an open-source tool, by Wada (2016) for annotating the UI screenshots.

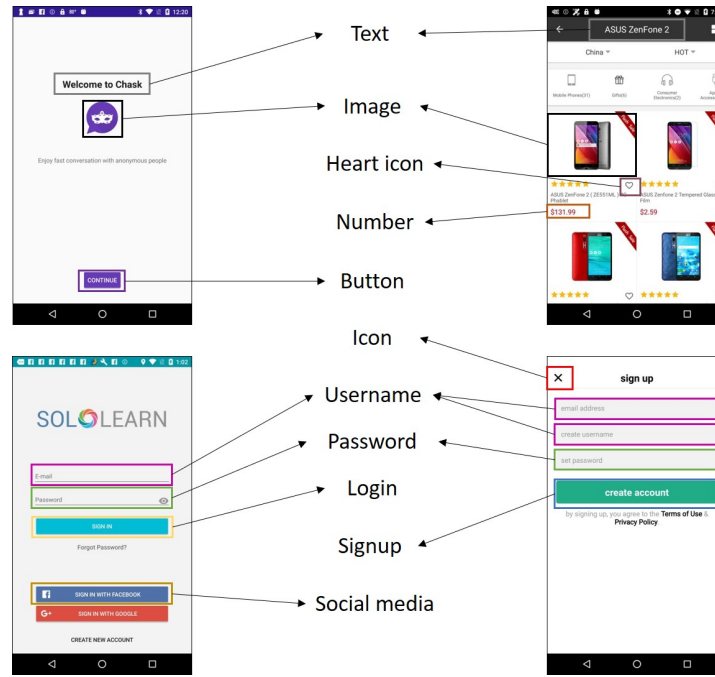
As a baseline, from each UI screenshot's view hierarchy, we extracted the leaf node UI elements and generated a LabelMe specific annotation JSON file. We then manually refined the bounding box positions, dimensions, groups and tagged the semantic UI element categories with this annotation file as a baseline. In the end, we converted the annotations from LabelMe JSON format to CSV and COCO JSON format for easy use with any open-source DNN models.

With this annotation process, we identified 100 unique UI elements. Figure 7.2 shows examples of some of the most common UI elements from the Wired dataset. The exhaustive list of all 100 categories of semantic UI elements from the Wired dataset is listed in table b.1.

*We extracted information from RICO dataset annotations and then manually refined them by adding semantic information to them.*

**Table 7.2:** Mean, standard deviation and mode of the number of UI elements per UI screenshot in each UI design pattern category

UI Design Pattern Category	Mean	Standard Deviation	Mode
Splash screen	6.23	2.52	5
Login	11.13	4.42	9
Account registration	11.79	4.35	10
Product catalog	35.07	12.95	31
Product page	17.54	6.36	17



**Figure 7.2:** Sample annotations with most common categories of semantic UI elements

#### 7.4 COLLECTED DATA

With the annotation files, we created the Wired dataset. This dataset contains semantic annotations of 2,751 UI screenshots. These semantic annotations correspond to the UI wireframe layouts. To create this dataset, we categorized UI screenshots from the RICO dataset into 5 UI design patterns and manually annotated them with 100 semantic categories of UI elements. On average, each UI screenshot contains 11.96 different categories of semantic UI elements. Table 7.2 shows exploratory data analysis of number of UI element labels per UI screenshot in each UI design pattern category.

#### 7.5 BENEFITS & APPLICATIONS

This dataset explores the context and nature of constituent UI element categories present in a UI screenshot. This dataset provides a basis for training various DNN networks in the following application domains.

**UI WIREFRAME GENERATION** Recently, few projects attempt to generate UI wireframes using DNNs. Most of these projects use Generative

Adversarial Networks (GAN) for this task. However, these projects use UI annotations without rich semantic information for training these models. The Wired dataset would provide an excellent base for training such models as it contains detailed semantic information of five different UI design patterns.

**UI LAYOUT REFINEMENT** UI layout refinement research attempts to automatically align and group UI elements present in a UI wireframe. As the Wired dataset contains grouping information in addition to the semantic annotation of constituent UI elements in a UI screenshot, this grouping information has the potential as ground truth for understanding the UI layouts and training such UI layout refinement networks.

## 7.6 SUMMARY

This chapter introduced the Wired dataset, our final dataset contribution. This dataset contains semantic annotations of 2,751 UI screenshots. To create this dataset, we selected UI screenshots from the RICO dataset, categorized them into 5 UI design patterns, and manually annotated them with 100 semantic categories of UI elements. We scoped this dataset to contain UI screenshots from e-commerce applications and categorized them into five different UI design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page. This detailed semantic information allows us to capture and understand the intricacies of UI wireframe layouts and also provides an excellent base for training powerful DNN models to infer information from the UI wireframes.





## PART II

# FORMATIVE ANALYSIS

In the upcoming chapters, we explain our two-part formative study in which we measured computational accuracy and human recognition accuracy of UI element sketches. The main focus of this analysis is to understand how humans and machine perceive UI sketches and determine whether the current state-of-the-art DNN models can augment UI/UX designers. To achieve this, we conducted a perceptual study with 76 UI/UX designers and compared their UI element sketch recognition accuracy with 26 trained UI element sketch classification DNN models. With these two study results, we further discuss and compare the differences in how well humans can recognize UI element sketches in contrast with machines' performance.



---

## HUMAN RECOGNITION STUDY

---

In this chapter, we discuss the first part of our two-part formative study. From our literature review, we identified there had been no controlled studies that compare differences in how much humans can recognize UI element sketches in contrast with the performance of machines. To bridge this gap, we conducted a perceptual study to measure the UI element sketch recognition accuracy of UI/UX designers, using the UISketch dataset ([Chapter 4](#)). This study aims to answer the following questions: given a UI element sketch, what is the accuracy with which UI/UX designers correctly recognize its category? Are there any UI element categories that are easier or harder to determine for UI/UX designers? This study provides us with an essential human baseline (specifically UI/UX designers) that we later compare with the computational recognition results. This chapter was published as a part of our research paper, “UISketch dataset” (Pandian, Suleri, and Jarke, [2021b](#)).

*We conducted a perceptual study to measure the UI element sketch recognition accuracy of UI/UX designers, using the UISketch dataset.*

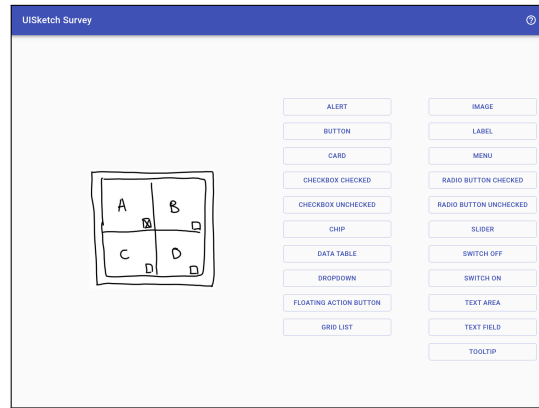
### 8.1 PARTICIPANTS

Using purposive and snowball sampling, we recruited 76 UI/UX designers for our study (M=35, F=41). The participants were between 23 to 38 years of age (M=30.61, SD=4.5) and had 1 - 13 years (M=4.2, SD=3.1) of prior prototyping experience. We made sure that these participants had not taken part in the UISketch dataset data collection. These participants belonged to five different countries: Germany, India, Pakistan, United Kingdom, and the U.S.A. Participants were compensated for their participation.

*76 UI/UX designers from 5 countries with 1 - 13 years of prior prototyping experience*

### 8.2 MEASUREMENTS

During this study, we aimed to measure the accuracy of recognition of UI/UX designer for any given UI element sketch. The independent vari-



**Figure 8.1:** Screenshot of the web application to collect data for human recognition. We show a random UI element sketch from our UISketch dataset on the left and a list of 21 UI element categories on the right-hand side

ables are the different sketches provided to the participants for each UI element category from the UISketch dataset, and the dependent variable is the accuracy, precision, and recall of human recognition.

### 8.3 APPARATUS

*Study was conducted remotely using a web application*

To measure the accuracy of recognition of UI elements by UI/UX designers, we created a web application<sup>1</sup> similar to the data collection questionnaire (Figure 8.1). This web application shows a random UI element sketch from our UISketch dataset on the left and a list of 21 UI element categories on the right-hand side. Participants could only choose one UI element category per any given sketch. The study took approximately 5 minutes. However, participants were allowed to take more time, if need be.

We forwarded the link to this web application to the participants and stored the results in our server. Participants filled out the questionnaire using the web application on their preferred system.

### 8.4 PROCEDURE

*Participants were shown a UI element sketch and asked to choose the appropriate UI element category*

The web application starts by requesting participants to provide informed consent. Once the participant agrees to the consent form, the web application creates a secure key for authentication and stores it in their browser's internal database. This step restricts the participants from taking part in

<sup>1</sup> <https://uisketch-survey.web.app/>



the study multiple times. It also helps in restoring the web application from the point they left off in case of internet disconnection.

Next, the participants are shown a randomly chosen UI element sketch from the UISketch dataset displayed on the left side of the screen, and a list of 21 UI element categories on the right side of the screen. Once the participant selects a particular UI element category, he can move on to the next question. The order in which UI element sketches are displayed is randomized. Each participant is provided with 21 UI element sketches to recognize.

## 8.5 ANALYSIS

To analyze the results of this study, we created a confusion matrix (a.k.a contingency table) to present the prediction results of human recognition in a clear and unambiguous manner. Here, by humans, we specifically refer to UI/UX designers.

Additionally, based on the collected data, we calculated the accuracy of human recognition. Accuracy represents the fraction of predictions that humans recognized correctly. We also calculated the precision to depict the exactness of human recognition. Precision represents the proportion of positive identifications that were actually correct. In addition to precision, we calculated the recall to depict the completeness of human recognition. Recall represents the proportion of actual positives that were identified correctly. Lastly, we calculated the F1 Score (a.k.a F Measure) that represents the balance between the precision and the recall.

We additionally measured the Pearson correlation coefficient between the accuracy of human recognition and the prior experience of our participants.

## 8.6 RESULTS & DISCUSSION

UI/UX designers recognized on average, **96.49%** of all UI element sketches correctly. We observed a minimal variance in human recognition accuracy over the 21 UI element categories. All the 76 participants correctly identified all occurrences of "Image", "Radio button checked/unchecked", "Checkbox checked" and "Slider". Whereas, the "Floating action button" category was recognized with the least accuracy of 92% (Table 8.1, Figure 8.2). However, there is no evidence that there is correlation based on our

*We analyzed the results of this study using accuracy, precision, recall, and F1 score. Also, we visualized the results using a confusion matrix*

*UI/UX designers recognized on average, **96.49%** of all UI element sketches correctly. Their recognition errors are usually caused due to the confusion between semantically similar UI element categories.*

**Table 8.1:** Classification report of human detection accuracy, precision, recall, and F1-score

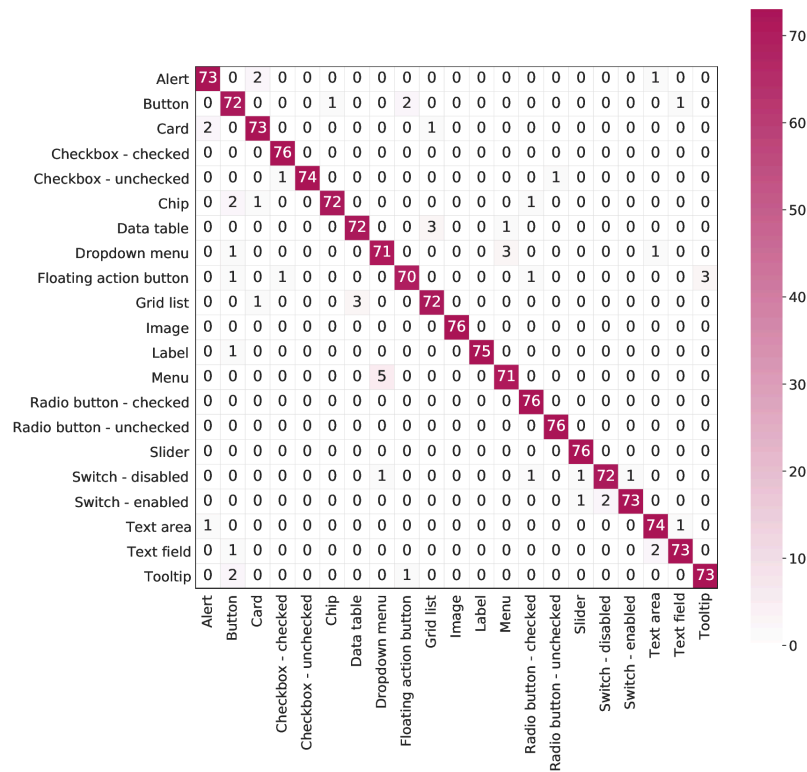
UI element category	Accuracy	Precision	Recall	F1-Score
Alert	96.05%	96.05%	96.05%	96.05%
Button	94.74%	90.00%	94.74%	92.31%
Card	96.05%	94.81%	96.05%	95.42%
Checkbox - checked	100.00%	97.44%	100.00%	98.70%
Checkbox - unchecked	97.37%	100.00%	97.37%	98.67%
Chip	94.74%	98.63%	94.74%	96.64%
Data table	94.74%	96.00%	94.74%	95.36%
Dropdown	93.42%	92.21%	93.42%	92.81%
Floating action button	92.11%	95.89%	92.11%	93.96%
Grid list	94.74%	94.74%	94.74%	94.74%
Image	100.00%	100.00%	100.00%	100.00%
Label	98.68%	100.00%	98.68%	99.34%
Menu	93.42%	94.67%	93.42%	94.04%
Radio button - checked	100.00%	96.20%	100.00%	98.06%
Radio button - unchecked	100.00%	98.70%	100.00%	99.35%
Slider	100.00%	97.44%	100.00%	98.70%
Switch - disabled	94.74%	97.30%	94.74%	96.00%
Switch - enabled	96.05%	98.65%	96.05%	97.33%
Text area	97.37%	94.87%	97.37%	96.10%
Text field	96.05%	97.33%	96.05%	96.69%
Tooltip	96.05%	96.05%	96.05%	96.05%

experiments ( $p=0.0636$ ) between the accuracy of human recognition and the prior experience of our participants.

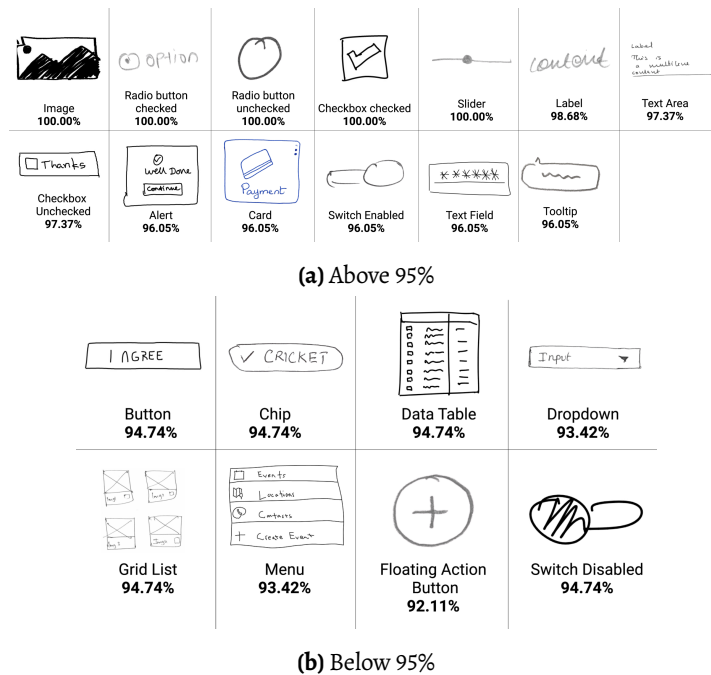
We have visualized the UI element categories with the highest human recognition in Fig. 8.3a, and the ones with the lowest human recognition in Fig. 8.3b.

We observed that human recognition errors are usually caused due to the confusion between semantically similar UI element categories. Here, semantic similarity refers to the likeness of the essence and purpose of the UI elements. Table 8.2 and Figure 8.4 show the UI element categories that are most often confused due to their semantic similarities.

However, structural similarity also accounts for some misrecognition occurrences. Here, structural similarity refers to the likeness of the shape and anatomy of the UI elements. Table 8.3 and Figure 8.5 show the UI element categories that are most often confused due to their structural similarities.



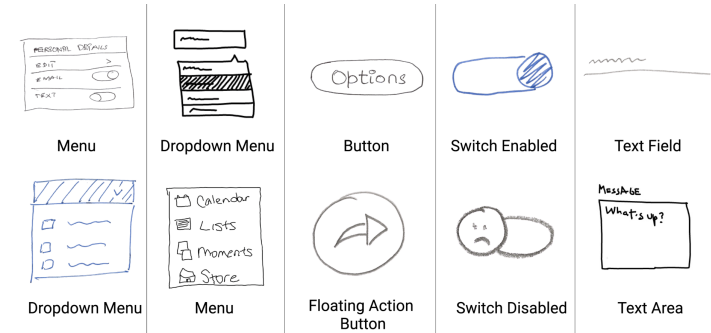
**Figure 8.2:** Confusion matrix for UI element categories on human recognized category vs the original category



**Figure 8.3:** Representative sketches from 21 UI element categories with highest (left) and lowest (right) human recognition accuracy

**Table 8.2:** Human detection accuracy for semantically similar UI elements

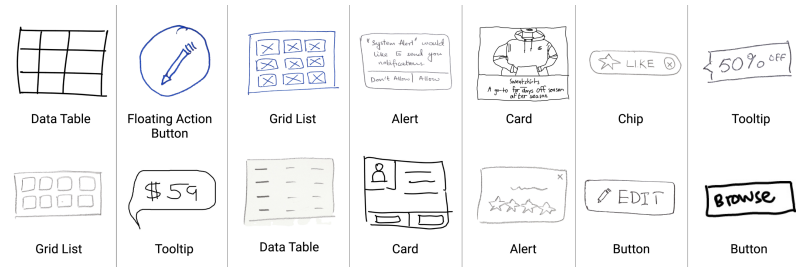
Displayed UI element category	Chosen UI element category	Percentage
Menu	Dropdown menu	6.58%
Dropdown menu	Menu	3.95%
Button	Floating action button	2.63%
Switch - enabled	Switch - disabled	2.63%
Text field	Text area	2.63%



**Figure 8.4:** UI element categories confused by UI/UX designers due to their semantic similarities

**Table 8.3:** Human detection accuracy for structurally similar UI elements

Displayed UI element category	Chosen UI element category	Percentage
Data table	Grid list	3.95%
Floating action button	Tooltip	3.95%
Grid list	Data table	3.95%
Alert	Card	2.63%
Card	Alert	2.63%
Chip	Button	2.63%
Tooltip	Button	2.63%



**Figure 8.5:** UI element categories confused by UI/UX designers due to their structural similarities

## 8.7 SUMMARY

This chapter discussed the first part of our formative study. In this perceptual study, we used the UI element sketches from the UISketch dataset and measured the accuracy with which UI/UX designers can recognize the UI element sketches. We conducted this study with **76 UI/UX designers** and found out that designers recognize UI element sketches with **~96% accuracy**. Further, we observed that human recognition errors are commonly caused due to the confusion between semantically similar UI element categories. We want to establish a baseline human through this formative study and, in the next chapter, compare it with the computer recognition accuracy of UI element sketches.



---

## COMPUTER RECOGNITION STUDY

---

This chapter summarizes the experiment and results of the second part of our two-part formative study. In this experiment, to contrast with the human recognition performance, we measured the accuracy of UI element recognition of the current state-of-the-art DNNs. In the following sections, we discuss the the DNN training pipeline, and its results. This chapter was published as a part of our research paper, “UISketch dataset” (Pandian, Suleri, and Jarke, [2021b](#)).

### 9.1 DNN MODELS

In this study, we measured the UI element recognition accuracy of DNN models. To measure this, we trained 26 classification neural networks, including image classification neural networks and sketch classification neural network that perform the classification tasks with high accuracy ([Table 9.1](#)). We chose only one sketch classification neural network, created by Seddati et al. ([2015](#)), because the other state-of-the-art sketch classification neural networks require temporal stroke information for training their models. However, this information cannot be collected in case of paper sketches, which is a preferred medium of UI sketching of a number of UI/UX designers. Therefore, we chose a model that can extract features from bitmap images.

To train and evaluate these 26 image and sketch classification neural networks, we used the PyTorch library created by Paszke et al. ([2019](#)) and fastai library created by Howard et al. ([2020b](#)). We then created a pipeline to preprocess the UI element sketches and split them for training (80%), validation (10%) and testing (10%). We used the training and validation datasets to train the classification models with suitable hyperparameters and finally evaluated the models with the test dataset. The following sections explain this process in detail.

*We trained 26 classification neural networks, including image classification neural networks and sketch classification neural network to measure the computational recognition accuracy of UI element sketches*

## 9.2 PREPROCESSING

*We preprocessed the UI element sketches by applying binary thresholding using OTSU, resizing them to a tensor of 3x224x224, and adding data augmentations for training the DNNs*

The UISketch dataset contains sketches from paper and digital questionnaires. The UI element sketches from this dataset are bitmap images with three channels (RGB) but with varying width and height. After several empirical experiments, we adapted the preprocessing technique recommended by Seddati et al. (2015) to acquire better training results.

To preprocess each image containing a UI element sketch, we resized it to 180x180 dimensions while preserving the aspect ratio to prevent the UI element sketch from getting distorted. Then we applied binary thresholding with automatic detection of thresholding value using OTSU thresholding method introduced by Otsu (1979). This step converts any value below the OTSU threshold value to 0 and above to 1 for each channel. Then we padded the image to resize it to 224x224 dimensions and inverted them. We represented these images in PyTorch as a tensor with shape as 3x224x224 dimensions. We then resized these tensors to suitable dimensions and normalized them based on the input requirements of each neural network. This preprocessing allows the sketches to be augmented within the frame for training while preserving all the essential features.

During the training epochs, we applied certain image transformations for augmenting the data. Data augmentation allows machines to learn better by allowing us to transform the images by scaling, shifting and rotating them. We applied the following data augmentations:

- Rotation: Random between -10 to 10 degrees
- Scaling: Random between 90% to 110%
- Perspective warping: Random between 0 to 0.2
- Normalization with imagenet stats

## 9.3 TRAINING

*We modified the hyperparameters for each model and fine-tuned them to obtain an optimum classification accuracy*

For training the neural networks, we split the dataset into training (80%), validation (10%) and testing (10%). We loaded the training and validation data to the GPU and applied the preprocessing and data augmentation strategies discussed in the previous section.

For each model in the list of 26 image and sketch classification models, we loaded the model with pretrained weights, training and validation data. Each model has a body (backbone or feature extractor) and a head



(classifier). To find the optimum learning rate, we followed the strategies advised by Howard et al. (2020a) and I. Goodfellow et al. (2016). We used the learning rate finder to obtain a suitable learning rate. Learning rate finder, proposed by Smith (2017), provides a methodology for estimating the appropriate learning rate for training neural networks without performing numerous experiments. Using this estimated learning rate curve provided by the learning rate finder, we chose the optimum learning rate where the loss curve is on a downward slope and approaches the minimum loss. We used this learning rate and trained each model with their feature extractor layers frozen for few epochs with the learning rate. This step allows the classification head of the model to be trained before we train the feature extractor.

Once the aforementioned preliminary training step of the neural network was completed, we unfroze the model to make all the layers trainable. Then we again ran the learning rate finder to find the optimum learning rate. Nevertheless, we also performed several empirical experiments on each model to find their appropriate learning rate and other hyperparameters, instead of merely relying on the learning rate finder. We trained each model until the training and validation loss values plateaued to prevent overfitting.

After training each model, we stored the trained model along with its weights as a PyTorch binary file. These trained models with weights can be loaded and reused for UI element sketch classification. We are also open-sourcing the trained models along with the scripts for reproducibility and replicability of this research and for further research applications.

#### 9.4 EVALUATION

We evaluated each trained classification model using the test dataset. This test dataset contains 10% of the UISketch dataset and does not contain any images from the training or validation datasets. We applied the pre-processing steps to the test dataset without data augmentation.

We used each model one by one to predict the UI element category of each image in the test dataset. Similar to the human recognition study, we then compared the predictions made by each model with the ground truth labels of test dataset using confusion matrix (Figure 9.1) and classification report (Table 9.2). As a result, we calculated the accuracy, precision, recall, and F1-Score, which we report in the next section.

*We evaluation each model with 10% of the UISketch dataset*

**Table 9.1:** UISketch test dataset prediction accuracy and top 5 accuracy of each computational classification models along with their published year and total number of parameters

Model name	Year	Parameters	Accuracy	Top 5 accuracy
ResNet 152 (He et al., 2015)	2015	60,261,461	91.77%	99.50%
ResNet 101 (He et al., 2015)	2015	44,617,813	90.60%	99.67%
DenseNet 201 (G. Huang et al., 2018)	2018	20,078,997	90.49%	99.39%
VGG 19 (Simonyan et al., 2015b)	2015	20,574,037	90.38%	99.50%
DenseNet 169 (G. Huang et al., 2018)	2018	14,207,381	89.93%	99.44%
Wide ResNet (Zagoruyko et al., 2017)	2017	68,951,893	89.93%	99.56%
GoogLeNet (Szegedy, W. Liu, et al., 2014)	2014	6,664,885	89.38%	99.33%
SqueezeNet v1 (Iandola et al., 2016)	2016	1,274,069	89.15%	99.22%
VGG 11 (Simonyan et al., 2015b)	2015	9,764,629	89.10%	99.28%
SqueezeNet v1.1 (Iandola et al., 2016)	2016	1,261,141	88.93%	98.94%
DenseNet 121 (G. Huang et al., 2018)	2018	8,018,837	88.93%	99.33%
ResNet 50 (He et al., 2015)	2015	25,625,685	88.88%	99.50%
EfficientNet-B2 (Tan et al., 2020)	2020	9,160,727	88.77%	99.22%
VGG 13 (Simonyan et al., 2015b)	2015	9,949,525	88.71%	99.33%
VGG 16 (Simonyan et al., 2015b)	2015	15,261,781	88.71%	99.28%
EfficientNet-ES (Tan et al., 2020)	2020	5,485,541	88.32%	99.39%
EfficientNet-B1 (Tan et al., 2020)	2020	7,841,333	88.10%	99.39%
ResNeXt (Xie et al., 2017)	2017	25,097,557	88.04%	99.39%
ResNet 34 (He et al., 2015)	2015	21,823,317	87.93%	98.94%
EfficientNet-B0 (Tan et al., 2020)	2020	5,335,697	87.93%	99.28%
EfficientNet-B3 (Tan et al., 2020)	2020	12,287,549	87.43%	99.11%
MixNet-L (Tan et al., 2019)	2019	7,383,569	87.32%	99.22%
ResNet 18 (He et al., 2015)	2015	11,715,157	87.04%	98.72%
AlexNet (Krizhevsky, 2014)	2014	2,745,173	86.26%	99.00%
Inception v3 (Szegedy, Vanhoucke, et al., 2015)	2015	5,335,697	85.71%	98.89%
DeepSketch (Seddati et al., 2015)	2015	16,912,917	81.87%	97.16%

## 9.5 RESULTS & DISCUSSION

ResNet 152 model achieves the highest accuracy of 91.77%. We observed that computational recognition errors are caused due to the structural similarity between UI element categories.

Table 9.1 shows the accuracy and top 5 accuracy<sup>1</sup> of 26 image and sketch classification neural networks. Among these 26 models, **ResNet 152 model** achieves the highest accuracy of **91.77%**. Whereas, DeepSketch model recognized UI element sketches with the least accuracy (81.87%).

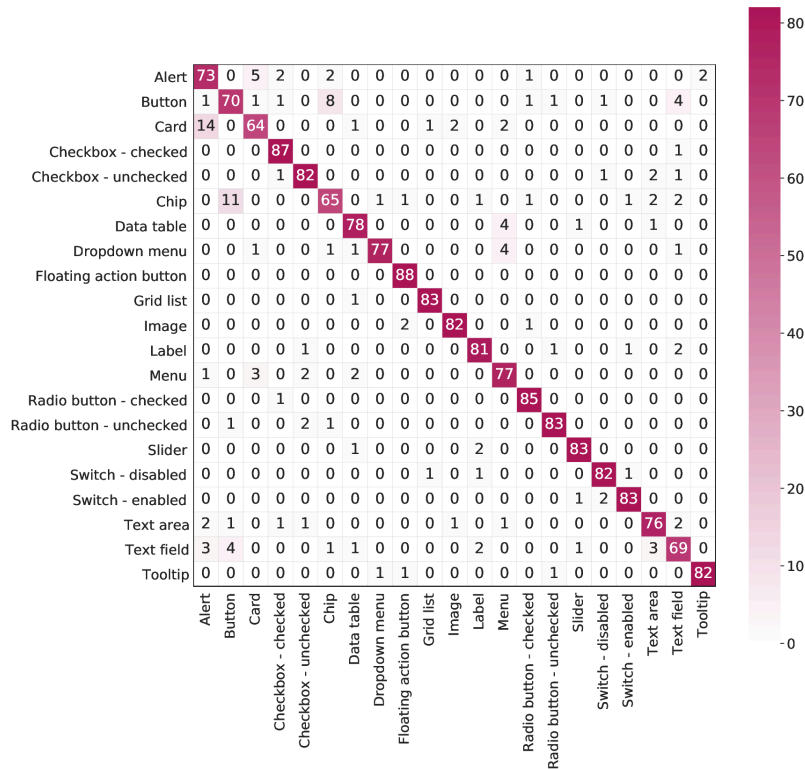
As mentioned earlier, the ResNet 152 model recognized, on average, 91.77% of all UI element sketches correctly. We observed that the model

<sup>1</sup> Top 5 accuracy is a measure to depict how often the predicted UI element category falls in the top 5 values predicted by the model

prediction shows 7.47 standard deviation in recognition accuracy over the 21 UI element categories. This model correctly identified all occurrences of "Floating action button". Whereas, the "Card" category was recognized with the least accuracy of 76.19%.

To understand the impact of recognition accuracy of deep learning models for sketches created by experienced professionals vs. other participants, we split our evaluation dataset further into sketches drawn by experienced professionals and others. As we mentioned in the paper (section 6), we used 10% of the collected sketches (1,798 sketches) as a test dataset for evaluation. We further split this test dataset into 788 sketches made by UI/UX designers & developers and 1,010 sketches drawn by other participants. We evaluated each of these datasets separately with the best-trained model, Resnet 152. From this experiment, we identified that Resnet 152, which gave 91.7% accuracy on the entire evaluation dataset, provides 92.76% accuracy for sketches drawn by UI/UX designers and developers. In contrast, it provides 90.99% accuracy for sketches drawn by other participants.

We further manually analyzed the sketches of both user groups. Based on this analysis, we speculate that the model provides better accuracy for



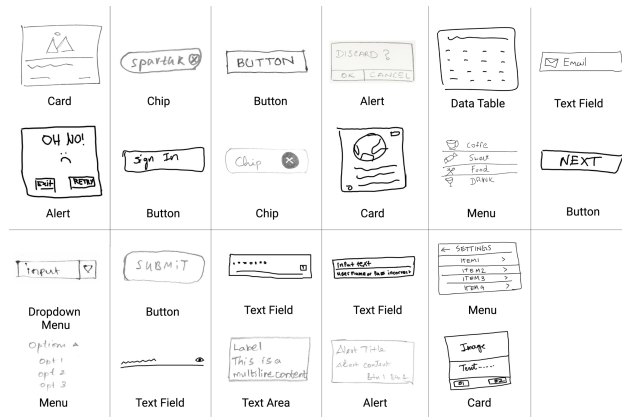
**Figure 9.1:** Confusion matrix for UI element categories on ResNet 152 predictions vs the ground truth

**Table 9.2:** Classification report of ResNet 152 model's accuracy, precision, recall, and F1-score

UI element category	Accuracy	Precision	Recall	F1-Score
Alert	85.88%	77.66%	85.88%	81.56%
Button	79.55%	80.46%	79.55%	80.00%
Card	76.19%	86.49%	76.19%	81.01%
Checkbox - checked	98.86%	93.55%	98.86%	96.13%
Checkbox - unchecked	94.25%	93.18%	94.25%	93.71%
Chip	76.47%	83.33%	76.47%	79.75%
Data table	92.86%	91.76%	92.86%	92.31%
Dropdown menu	90.59%	97.47%	90.59%	93.90%
Floating action button	100.00%	95.65%	100.00%	97.78%
Grid list	98.81%	97.65%	98.81%	98.22%
Image	96.47%	96.47%	96.47%	96.47%
Label	94.19%	93.10%	94.19%	93.64%
Menu	90.59%	87.50%	90.59%	89.02%
Radio button - checked	98.84%	95.51%	98.84%	97.14%
Radio button - unchecked	95.40%	96.51%	95.40%	95.95%
Slider	96.51%	96.51%	96.51%	96.51%
Switch - disabled	96.47%	95.35%	96.47%	95.91%
Switch - enabled	96.51%	96.51%	96.51%	96.51%
Text area	89.41%	90.48%	89.41%	89.94%
Text field	82.14%	84.15%	82.14%	83.13%
Tooltip	96.47%	97.62%	96.47%	97.04%

sketches drawn by experienced professionals as they follow some drawing conventions for sketching UI elements, as mentioned in section 4.1. In contrast, the other participants tend to replicate in detail the sample images provided during the data collection study. Therefore, the model finds it challenging to recognize UI element sketches with more information such as entirely written text and replicated images drawn by other participants instead of squiggly lines and rectangles with crosses drawn by experienced professionals.

We observed that computational recognition errors are solely caused due to the confusion between the structural similarity between UI element categories. Here, structural similarity refers to the likeness of the shape and anatomy of the UI elements. [Table 9.3](#) and [Figure 9.2](#) show the UI element categories that are most often confused due to their structural similarities.



**Figure 9.2:** UI element categories with structural similarities

**Table 9.3:** Computational detection accuracy for structurally similar UI elements

Ground truth	Predicted by ResNet152	Percentage
Card	Alert	16.67%
Chip	Button	12.94%
Button	Chip	9.09%
Alert	Card	5.88%
Data table	Menu	4.76%
Text field	Button	4.76%
Dropdown menu	Menu	4.71%
Button	Text field	4.55%
Text field	Text area	3.57%
Text field	Alert	3.57%
Menu	Card	3.53%

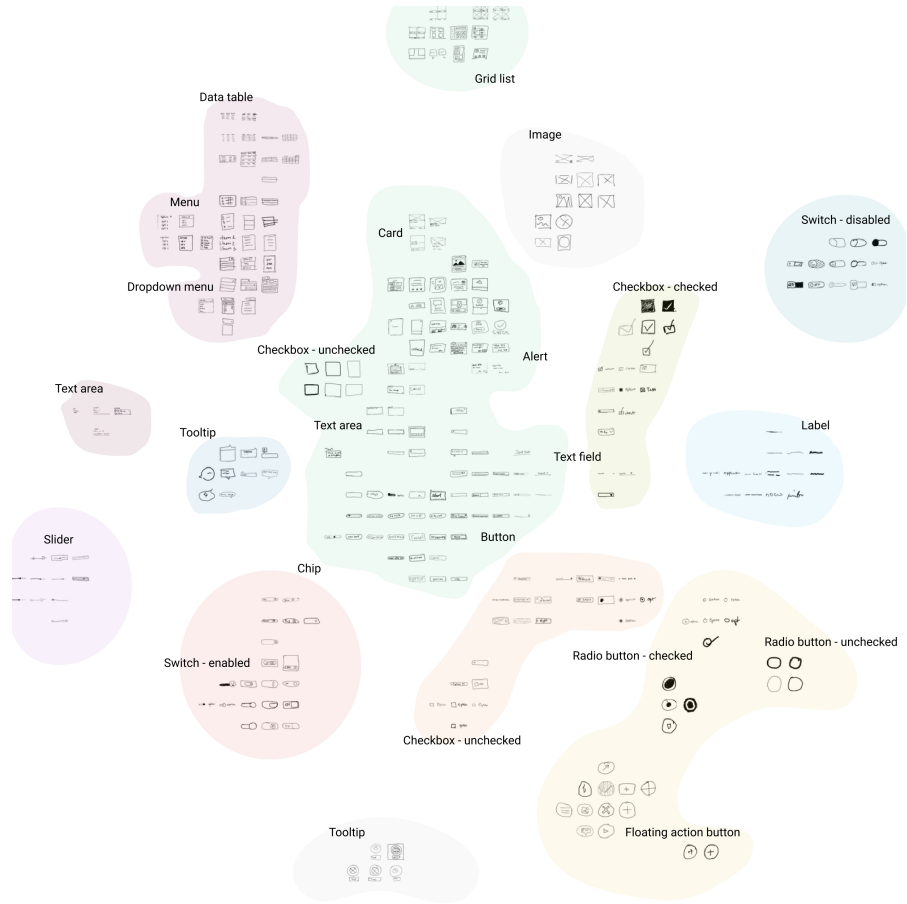
## 9.6 HUMAN VS COMPUTER RECOGNITION

Based on the results of the recognition, we identified that the humans and computer have a similar pattern of confusing UI elements with structural similarities, e.g., "Card" with "Alert" and "Chip" with "Button".

However, there are a few UI elements that are confused by computational recognition due to their structural similarities, but humans confuse them due to their semantic similarities, e.g., "Dropdown menu" with "Menu", and a "Text field" with "Text Area".

Since computational recognition is solely based on structural similarities, there are a few UI elements that are confused by the recognition model but are not confused by humans, e.g., a "Text field" with "Button", "Text Field" with "Alert", and "Menu" with "Card". On the contrary, humans confuse these above-mentioned UI elements with different UI elements

*Both humans and computers have a similar pattern of confusing UI elements with structural similarities. However, humans mostly confuse recognizing UI elements due to their semantic similarity*

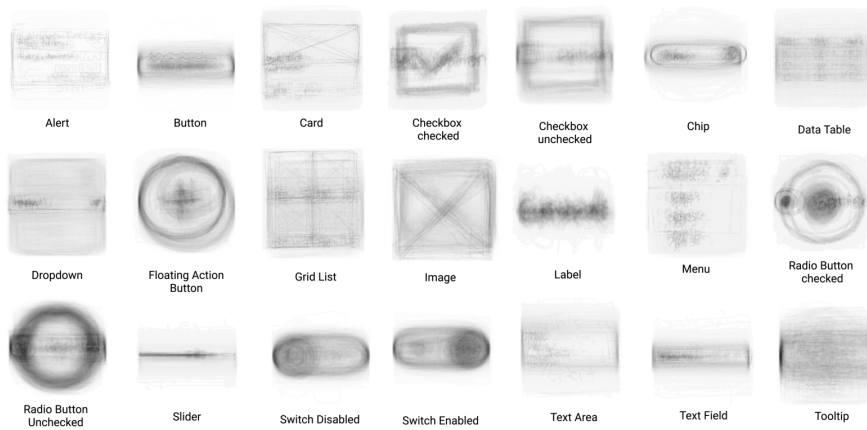


**Figure 9.3:** t-SNE visualization of representative sketches of 21 categories of UI elements from the test dataset of UISketch dataset. We have manually marked 13 clusters based on the proximity of these clusters.

based on their semantic similarities, e.g., a "Text field" with "Text Area" instead of "Button" or "Alert", "Button" with "Floating Action Button" instead of "Text field", and "Menu" with "Dropdown Menu" instead of "Card". We speculate that humans could identify specific UI elements better based on their context or given text content.

## 9.7 ANALYSIS OF UI ELEMENT SKETCHES

To further analyze the reasoning behind why the ResNet 152 model confuses few UI element categories, we used its trained feature extractor (backbone) and reduced the dimensionality of the test dataset images from  $3 \times 224 \times 224$  to  $1 \times 1024 \times 1024$ px. Then we used the t-SNE algorithm, proposed by van der Maaten et al. (2008), to reduce the features further and plotted them as a 2D graph (Figure 9.3). From this t-SNE plot, we can observe



**Figure 9.4:** Representative images showing the activated pixel areas and the underlying structure of all UI element categories.

that the computational classification model clusters the images based on their structural similarities. As expected, there is no confusion based on semantic similarities between UI elements; floating action button is farther away from the button.

To understand the structural information and pixel density of each UI element category, we visualized the sketches from each category of the UISketch dataset as a representative image. To generate these representative images, we preprocessed the UI element sketches as previously described in the preprocessing section. For each UI element category, we then normalized all the sketches and calculated the average value at each pixel. Next, we scaled the averaged sketches back to an RGB image with pixel values between 0 to 255 to obtain the representative image for each UI element category. [Figure 9.4](#) shows the activated pixel areas and the underlying structure of each UI element category.

## 9.8 SUMMARY

In this chapter, we described our training procedure and the results of our study on computer recognition of UI element sketches. In this study, we trained and evaluated 26 classification DNN models and understood that computational recognition errors are commonly caused due to the confusion between the structural similarity between UI element categories. This study further revealed that ResNet 152 model identifies UI element sketches from the test dataset with 91.77% accuracy and has not reached human-level performance yet. This formative study investigated the dif-

ference between human and computer recognition of UI element sketches. From this study, we conclude that although DNN models can recognise UI sketches, it still has not reached the high accuracy of UI/UX designers. Therefore, instead of replacing UI/UX designers, it is beneficial to automate the LoFi prototyping task to efficiently assist UI designers in their design process while maintaining the autonomy of UI designers. In the next part of this thesis, we discuss the first AI tool in the Blackbox Toolkit: Akin, a UI wireframe generator. This tool assists the UI designers before they start their LoFi prototyping process.



## PART III

# AKIN

In this part, we introduce Akin, a UI wireframe generator that assists UI designers *before LoFi prototyping*. Akin generates multiple UI wireframes for a given UI design pattern; thus, enabling designers to conceptualise UI designs quickly. We created Akin by modifying the SAGAN model and training it using a subset of the Wired dataset. Akin's generative model provides an Inception Score of 1.63 (SD=0.34) and a Fréchet Inception Distance of 297.19. Our user evaluation studies revealed that UI/UX designers considered wireframes generated by Akin to be as good as wireframes made by designers. Moreover, designers identified Akin-generated wireframes as designer-made 50% of the time. Additional studies with the ASQ to assess user satisfaction indicates that designers experience an above-average satisfaction level towards ease of task completion (4.6), time taken (5.7), and supporting information (5.0) upon utilising Akin before the LoFi prototyping process. Their qualitative feedback showed that both novice designers and experience designers found AI assistance helpful in addition to the traditional UI prototyping process.



<https://akin.blackbox-toolkit.com>

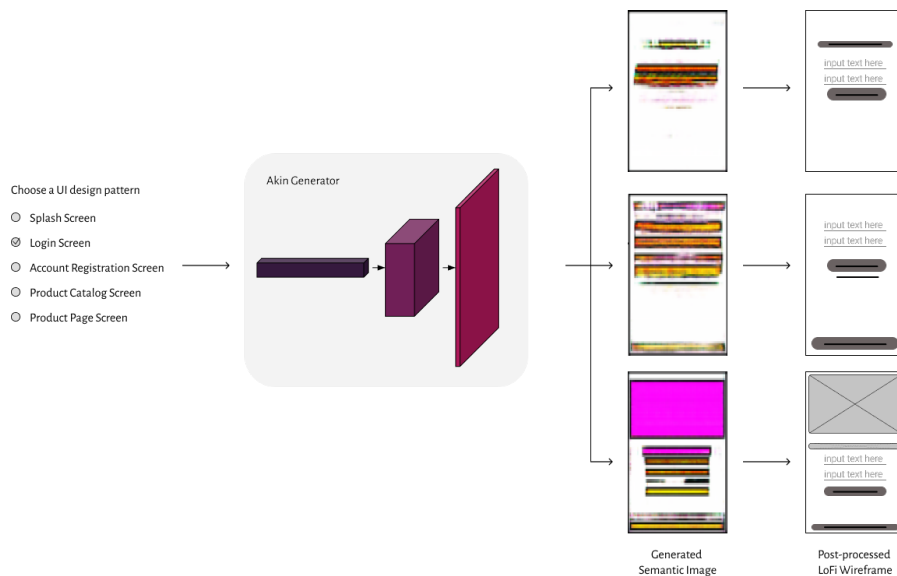


# 10

## BACKGROUND & PROPOSED SOLUTION

Akin is a UI wireframe generator that assists UI designers before the LoFi prototyping by allowing them to choose a UI design pattern and providing them with multiple UI wireframes for a chosen UI design pattern (Figure 10.1). This research aligns with the current UI design practice where, as (Herring et al., 2009; Suleri, Kipi, et al., 2019) claims, the UI designers search for possible design alternatives while creating a UI design, and through Akin as AI assistant, we provide these possible UI design alternatives to designers. This part of the dissertation was published at IUI 2021 (Gajjar et al., 2021) created under my supervision based on my research. This chapter briefly recaps the existing literature on UI layout generation, emphasises the research gap in them and presents our proposed solution.

*Akin assists designers **before** their LoFi prototyping task by generating wireframes for a given UI design pattern.*



**Figure 10.1:** Akin is UI wireframe generator that generates UI wireframes for the chosen UI design pattern

## 10.1 BACKGROUND

*Only three recent research projects focus on UI design and attempt to generate UI wireframes using DNNs.*

A considerable amount of literature has been published on generating and optimising document layouts; however, only three research projects focus on UI design and attempt to generate UI wireframes (Section 2.4). All the research projects that generate and refine UI wireframes use either just DNNs or a hybrid of DNNs along with classic pattern recognition algorithms. The initial proposal on this research domain was by Nguyen *et al.*. In their work-in-progress paper, Nguyen *et al.* proposed their vision of generating UI layouts using Generative Adversarial Network (GAN); however, they did not publish subsequently about their UI layout generator architecture or results. Later in 2019, J. Li *et al.* (2019) introduced LayoutGAN that generates graphic, document and UI wireframe layouts by modelling geometric relations of different types of 2D elements (such as UI elements) present in the content. The authors used the RICO dataset with 5 UI elements to show the use-case of LayoutGAN in generating UI wireframes. In the next year, Gupta *et al.* (2020) proposed LayoutTransformer, a auto-regressive self-attention network, to generate document and UI wireframes. In contrast with LayoutGAN, this model can take an empty or partial layout as input and generate realistic document or UI wireframe layouts. Recent research on UI wireframe generation is the Neural Design Network by Lee *et al.* (2020). The authors constructed Neural Design Network using a Graph Convolution Network (GCN) that generates UI wireframes as layout graphs based on user constraints.

## 10.2 IDENTIFIED GAPS

*All of these research projects were not evaluated by UI designers.*

The notable research gap in all three aforementioned research is that these research projects did not evaluate the quality of the generated UI wireframes by user studies. . Without such user evaluation with experts (here UI/UX designers), neither the efficacy nor the utility of the UI wireframe generation can be categorically proven. Another common research gap between all three research is that these research train their UI wireframe generator considering all UI designs are equal. However, UI wireframes can be further categorised into UI design patterns, and the layout differs vastly between two UI design patterns. UI design patterns describe a re-occurring problem in UIs and propose a solution to that problem. They are used as *lingua franca* to discuss UI design concepts; a given UI design can be categorised into a UI design pattern. Without such distinction,

any generated UI wireframe cannot be applied to a problem that the UI designer is currently solving. Therefore the generated designs from DNNs will be generic and cannot be used for a specific scenario.

### 10.3 PROPOSED SOLUTION

To overcome these research gaps, we propose Akin, a UI wireframe generator that generates multiple UI wireframes dynamically for a given UI design pattern. Akin assists designers before they start the UI design process by providing them with quick alternate designs for a given UI design pattern. The design flow by adding Akin is similar to the traditional UI design process where UI designers look for existing UI design examples for inspiration and reconstruct them as purported by (Herring et al., 2009; Suleri, Kipi, et al., 2019).

To create Akin, we experimented with several data representation of UI wireframe layouts and DNN models. Through these empirical experiments, we chose the SAGAN model by Zhang et al. (2019) for creating Akin. We modified the SAGAN model into a conditional SAGAN with embeddings and trained it to generate Android UI wireframes of a chosen UI design pattern. Akin's SAGAN model is trained using a subset of the Wired dataset (Chapter 7) with 500 UI screenshots of 5 design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page. To quantify the performance of Akin, To evaluate the SAGAN model, we used the Inception Score (IS) by Salimans et al. (2016) to measure the quality of machine-generated wireframes and Fréchet Inception Distance (FID) by Heusel et al. (2018) to measure the similarity between machine-generated and designer-made wireframes.

Further, to fix the research gap mentioned in the previous section, we evaluated the quality of Akin-generated wireframes with 15 UI/UX designers by conducting three quantitative and qualitative user evaluations. We evaluated the model with commonly used quantitative user evaluations to assess image generation models: rating-preference judgment and rapid scene categorisation. To answer RQ4, we conducted another quantitative study with 10 UI/UX designers using the After Scenario Questionnaire, a standard questionnaire to measure user satisfaction, along with qualitative semi-structured user interviews.

In the upcoming chapter, we discuss the experiments we conducted to finalise the final data representation of UI wireframe layouts and the model architecture of Akin.

**Model:** conditional  
Self-Attention  
Generative  
Adversarial Networks

**Dataset:** Subset of the  
Wired dataset  
containing 500 UI  
screenshots of 5 design  
patterns

**Evaluation:** Two AI  
metrics, three  
quantitative and one  
qualitative user  
evaluation with  
UI/UX designers



---

## MODEL ARCHITECTURE & DATA REPRESENTATIONS

---

Recent advancements in Deep Generative models paved the way for data generation and syntheses in various domains such as text, image, audio, and spatial domains. We utilized these recent advancements in AI research to create Akin, a UI layout generator. This chapter describes the experiments we conducted with different data representations of UI wireframe layouts and the different generative model architectures to create Akin. Detailed information of these experiments and results are published at Gajjar (2020) created under my supervision based on my research.

### 11.1 MODEL ARCHITECTURES

Generative models are a subcategory of unsupervised learning models. Foster (2019) defines that a generative model describes, in probabilistic terms, how a given training dataset is generated. After training such a generative model, we can extract samples from it to generate new data similar to the training data. Therefore, these generative models, in essence, learn and model a probability distribution. Deep generative models are DNNs that are modelled for generative tasks. For creating Akin, from our literature review on UI layout generation Section 2.4, we identified Generative Adversarial Networks (GANs) and autoregressive models for generating UI layouts.

*We experimented with four model architectures to create Akin*

#### 11.1.1 Generative Adversarial Networks

GANs are one of the popular deep generative models proposed by I. J. Goodfellow et al. (2014). It is made of two DNNs that are trained simultaneously: a generative model ( $\mathcal{G}$ ) and a discriminative model ( $\mathcal{D}$ ). During training, ( $\mathcal{G}$ ) is pitted against the ( $\mathcal{D}$ ) where  $\mathcal{G}$  learns to generate fake data that is a sample of the training data distribution and  $\mathcal{D}$  learns to

*Currently, GANs are one of the most popular deep generative models. They are trained using an adversarial process that corresponds to a minimax two-player game and improves both the models until the generated fake data is analogous to the training data.*

determine whether the generated sample is from the model distribution or the data distribution. This adversarial training process corresponds to a minimax two-player game and improves both the models until the generated fake data is analogous to the training data. These GANs can be made conditional by adding condition ( $\mathcal{C}$ ) as an input along with the noise vector for both  $\mathcal{G}$  and  $\mathcal{D}$ .

Since its inception in 2014, researchers have created several variations of GANs, such as Deep Convolutional GAN by Radford et al. (2016), Wasserstein GAN by Arjovsky et al. (2017), and Self-attention GAN (SAGAN) by Zhang et al. (2019), for different domains and use cases.

Unlike other variations, Self-attention GAN (SAGAN) combines the Self-Attention mechanism proposed by Vaswani et al. (2017) with DCGAN to capture long-range dependencies in training data. The authors claim that, by using self-attention, the discriminator can enforce geometric constraints globally; thus, it performs better for generating images with geometric patterns.

### 11.1.2 Autoregressive Models

Another approach in deep generative models is autoregressive DNNs. In general, an autoregressive statistical model predicts future values based on the given past values. Similarly, autoregressive DNN architectures use neural network architectures in autoregression task.

*Autoregressive models such as PixelCNN uses convolutional layers to predict the next pixel given a current context of neighbouring pixels*

Oord et al. (2016) proposed PixelCNN that uses such autoregressive connections to model images pixel by pixel. By this procedure, the authors decompose a given training data into a product of conditionals. PixelCNN uses Convolution Neural Networks (CNN) and specializes in image generation task. Salimans et al. (2016) simplified PixelCNN, improved its performance and proposed PixelCNN++. Unlike PixelCNN, their model uses a discretized logistic mixture likelihood on the pixels, among other modifications, to generate better quality images.

Similarly, Gupta et al. (2020) proposed another architecture using an auto-regressive self-attention network called LayoutTransformer to generate document and UI wireframes. This model uses Transformer-Encoder architecture to generate layouts. The novelty in this approach is that, once trained, this model can take an empty or partial layout as input and generate document or UI wireframe layouts.

The goal of Akin is to generate multiple UI wireframes for a given UI design pattern. Therefore, the generative models we create must be con-



ditional so that they can model a UI layout similar to the given UI design pattern. Therefore, we modified any chosen architecture into conditional generative models. The upcoming section describes the experiments we conducted for creating Akin with a modified conditional version of SAGAN, PixelCNN++, and transformer-encoder model architectures using two different data representations.

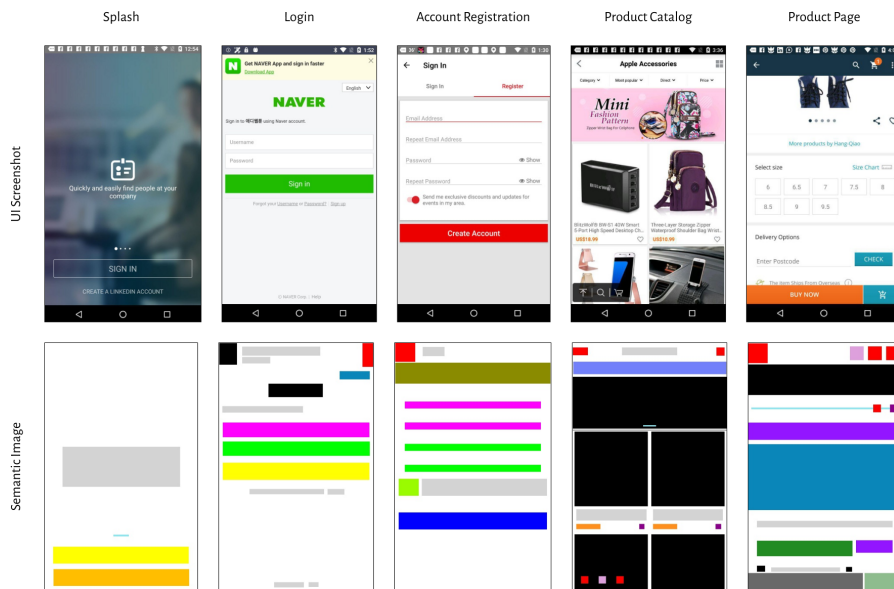
## 11.2 DATA REPRESENTATIONS

Apart from the differences in architectures in deep generative models, the data representation they use to train or evaluate also differs within them; thus, these models enable us to use them in different domains such as spatial, audio, and image. To address RQ2, we experimented by representing UI wireframe layout in spatial and image domain to train GANs, PixelCNN++, and transformer-encoder models.

*We experimented with two data representation of UI wireframe layouts to train and evaluate Akin*

### 11.2.1 Image Domain: Semantic Images

We extended the initial idea by T. F. Liu et al. (2018) on representing layout as semantic images. In their research, they created semantic images with two colours to represent textual and non-textual elements. They used this



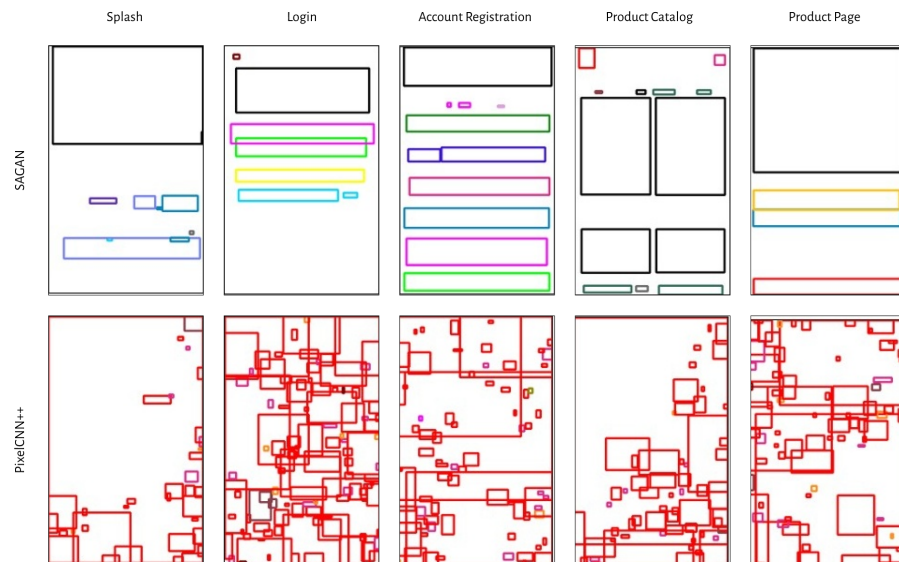
**Figure 11.1:** Sample UI screenshot for each of the 5 UI design pattern in the Wired dataset along with their semantic images

*We transform a UI wireframe annotation to a semantic image by selecting a unique colour to represent each UI element bounding box in the given wireframe*

data to successfully train a variational auto-encoder model to reduce the dimensions of a given UI screenshot.

Following their footsteps, we represent a UI wireframe layout as a semantic image with a unique colour representing each UI element bounding box. We used the Wired dataset of UI wireframe layouts [Chapter 7](#) to train Akin. This dataset contains 2,751 UI wireframes annotated with 100 different semantic UI element categories. We chose 500 UI wireframes of 5 UI design patterns with 28 commonly occurring semantic UI element categories in a UI wireframe layout from this dataset for training and evaluation. Therefore, to create a semantic image of these 500 UI wireframes, we sampled 28 unique colours as 3D vectors in an RGB colour space with near maximum possible L2 distance. In addition, we mapped the pair of colours with the largest L2 distance to UI elements that are commonly occurring near each other. Using this UI element - colour mapping, we converted the UI wireframe annotations to semantic images. [Figure 11.1](#) shows sample semantic images for different UI design patterns.

We used these semantic images of UI wireframes for training different model architectures such as SAGAN and PixelCNN++. We modified both these models into a conditional model, where generator ( $\mathcal{G}$ ) takes two inputs: category of UI design pattern and a noise vector, and discriminator ( $\mathcal{D}$ ) takes two inputs: category of UI design pattern and semantic image with 128x128 dimensions. From these training experiments, we identified that the SAGAN model performed better than the PixelCNN++ model,



**Figure 11.2:** Samples of semantic UI wireframes generated by trained SAGAN and PixelCNN++ model for each of the 5 UI design pattern

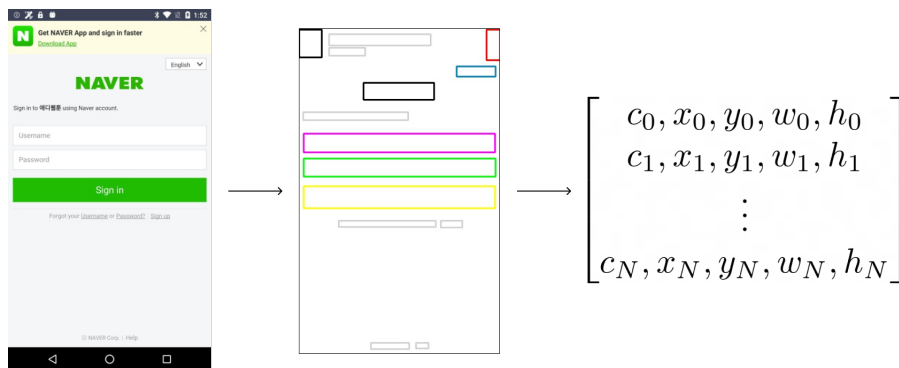
where the latter did not generate any meaningful semantic images. In contrast, the SAGAN model generated semantic image for a given UI design pattern with adequate quality. Figure 11.2 shows a samples of generated semantic UI wireframes by trained SAGAN and PixelCNN++ models. The issue with this approach is that the generated semantic images had colour leaking and merging between two spatially close UI elements. Therefore, we added an algorithmic post-processing step to identify the UI elements from the semantic images.

### 11.2.2 Spatial Domain: Layout Vectors

J. Li et al. (2019) proposed another approach of representing a UI wireframe layout as a vector list of constituent UI element categories and their corresponding location and dimension. Therefore, if a deep generative model learns this information and generates a similar representation, it can be easily converted back into a UI wireframe. However, one issue with this approach is that each UI wireframe can contain a different number of constituent UI elements. Therefore, to make the size of the input vector uniform, we have fixed a maximum number of UI elements per screen ( $N$ ) and fill the rest with empty vectors. As per our exploratory data analysis of the Wired dataset, we set  $N$  as 45, the maximum possible UI elements present in a screen in the dataset.

Similar to the previous data representation approach, we chose the same 500 images with 28 categories of UI elements. Using a script, we converted all the UI wireframe annotations to the layout vector representation. We padded the UI wireframes that contain a count of constituent UI elements less than  $N$  with an empty UI element category "Other", represented as 0.

*We represent UI wireframe layout as a series of vectors, each containing information of constituent UI element categories and their corresponding location and dimension*



**Figure 11.3:** Sample layout vector representation of a login UI design pattern

This layout vector contains all the constituent UI elements in sequence from top-left to bottom-right. Each constituent UI element and their bounding box is represented as  $\mathcal{V}$  as described in Equation 11.1. Thus, a UI wireframe is represented as  $[\mathcal{V}_0, \mathcal{V}_1, \dots, \mathcal{V}_N]$  as show in Figure 11.3.

$$\mathcal{V}_i = [c_i, x_i, y_i, w_i, h_i] \quad (11.1)$$

where:

$\mathcal{V}$  = representation of a constituent UI element in a UI wireframe

$c$  = one-hot vector of the category of UI element (0-28)

$x$  = left-most position of the bounding box

$y$  = top-most position of the bounding box

$w$  = width of the bounding box

$h$  = height of the bounding box

We used this layout vector representation of the UI wireframes to train a modified conditional transformer-encoder model. However, unlike generator models in GANs, which takes a noise vector, a class label as input, this model also requires the number of UI elements to generate in the UI wireframe as input. Moreover, this model did not generate meaningful layouts. Thus, the only advantage of this model over others is that it generates the UI wireframe in the spatial domain and therefore does not require any post-processing.



**Figure 11.4:** Samples of layout vector UI wireframes generated by trained Transformer-Encoder model for each of the 5 UI design pattern

Based on these empirical trials, we chose Self-Attention GAN (SAGAN) architecture for UI wireframe generation, which generated meaningful layouts. In the upcoming chapter, we describe the implementation details, configuration, and training process of the Akin: UI wireframe generator.

# 12

---

## IMPLEMENTATION

---

As discussed in the previous chapter, after several experiments with model architectures and data representations, we chose the Self-Attention Generative Adversarial Network (SAGAN) for creating Akin. Additionally, we modified the SAGAN model architecture and made it conditional using an embedding layer. This trained model generates a UI wireframe for a given UI design pattern. This chapter briefly explains the implementation steps in creating Akin.

### 12.1 IMPLEMENTATION APPROACH

Akin is a UI wireframe generator that aims to assist designers before they start their design process by generating multiple UI wireframes for a chosen UI design pattern. Therefore, we modified the SAGAN model architecture to accept an additional input ( $\mathcal{C}$ ) representing one of the five chosen UI design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page.

Upon literature review, we identified two possible alternatives of adding generation conditions to a GAN architecture: adding condition as additional dimension as proposed by Mirza et al. (2014), or adding an embedding layer of the condition then concatenating with input as proposed by Denton et al. (2015). After few empirical experiments, we chose to add an embedding layer to both generator ( $\mathcal{G}$ ) and discriminator ( $\mathcal{D}$ ) of the SAGAN model architecture to make it conditional.

We followed the traditional GAN model training pipeline outlined by I. Goodfellow et al. (2016) by training generator and discriminator in alternate steps then updating their respective gradients. For training the discriminator to classify real and fake semantic UI wireframe images, we used the Wired dataset (Chapter 7).

*From empirical experiments, we chose the Self-Attention Generative Adversarial Network (SAGAN) for creating Akin*

## 12.2 DATASETS

The Wired dataset contains 2,751 UI wireframes annotated with 100 different semantic UI element categories and classified into five commonly used UI design patterns in an e-commerce application: Splash Screen, Login, Account Registration, Product Catalog, and Product Page. To reduce class imbalance, we sampled 500 UI wireframes of 5 UI design patterns (100 for each UI design pattern) from the Wired dataset with 28 commonly occurring semantic UI element categories for training and evaluation.

*We trained Akin using 500 semantic images of 5 commonly used UI design patterns in an e-commerce application*

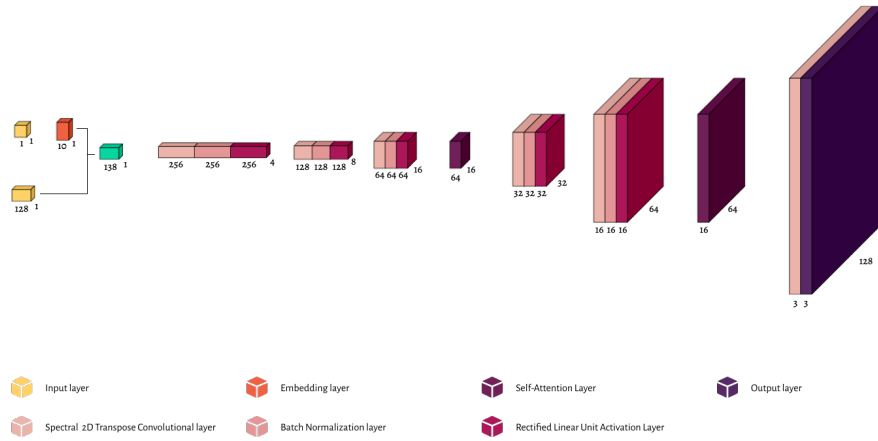
We chose to train Akin's SAGAN model in the image domain using semantic UI wireframe images from our prior experiments. We followed the steps outlined in the previous chapter ([Section 11.2.1](#)) to convert the 500 chosen UI wireframes from the Wired dataset into semantic images. Further, we resized these semantic images to 128x128x3 dimensions to train the discriminator model to distinguish real and fake semantic UI wireframe images ([Figure 12.2, A](#)).

## 12.3 IMPLEMENTATION DETAILS

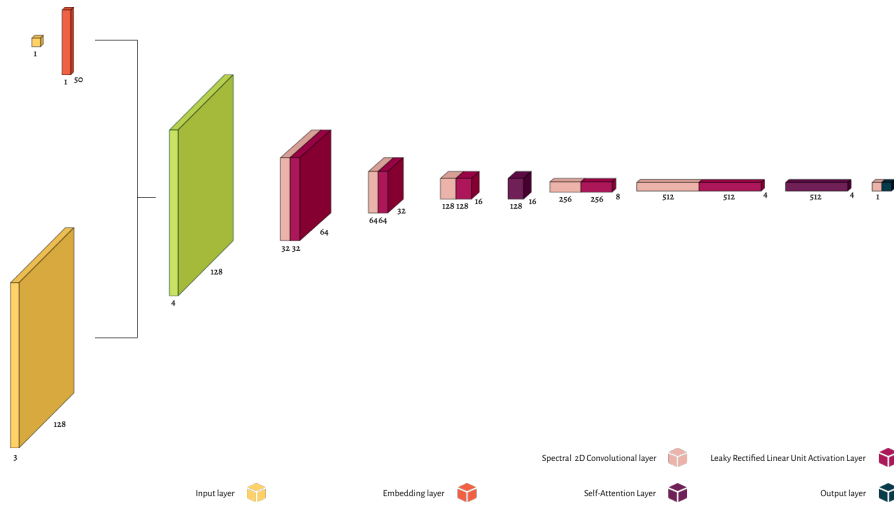
To create Akin, we implemented a conditional SAGAN model using Tensorflow 2.0 library created by Abadi et al. (2015). This model contains two neural networks: generator ( $\mathcal{G}$ ) and discriminator ( $\mathcal{D}$ ). [Figure 12.1](#) shows the Akin's model architecture of generator ([Figure 12.1a](#)) and discriminator ([Figure 12.1b](#)).

The generator ( $\mathcal{G}$ ) takes a random noise (size = 128) and an embedding of UI design pattern category (size = 10) as input ([Figure 12.2, B](#)). These inputs are then concatenated as passed to the hidden layers of the DNN. The hidden layers consist of 5 blocks, each containing three layers: a Spectral 2D transpose convolutional layer followed by a Batch Normalization layer and a Rectified Linear Unit (ReLU) activation layer. After the third and the fifth block, we add a Self-Attention layer. The final output layer is a Spectral 2D transpose convolutional layer which outputs a generated semantic image of UI wireframe ([Figure 12.2, C](#)).

The discriminator ( $\mathcal{D}$ ) takes an embedding of UI design pattern category (size = 50) and either a semantic image of UI wireframe from the training dataset (128x128x3) or a machine-generated image from the generator (128x128x3) as input. Similar to  $\mathcal{G}$ , these inputs are then concatenated and passed to the hidden layer in  $\mathcal{D}$ . These hidden layers consist of 5 blocks, each containing two layers: a Spectral 2D convolution layer



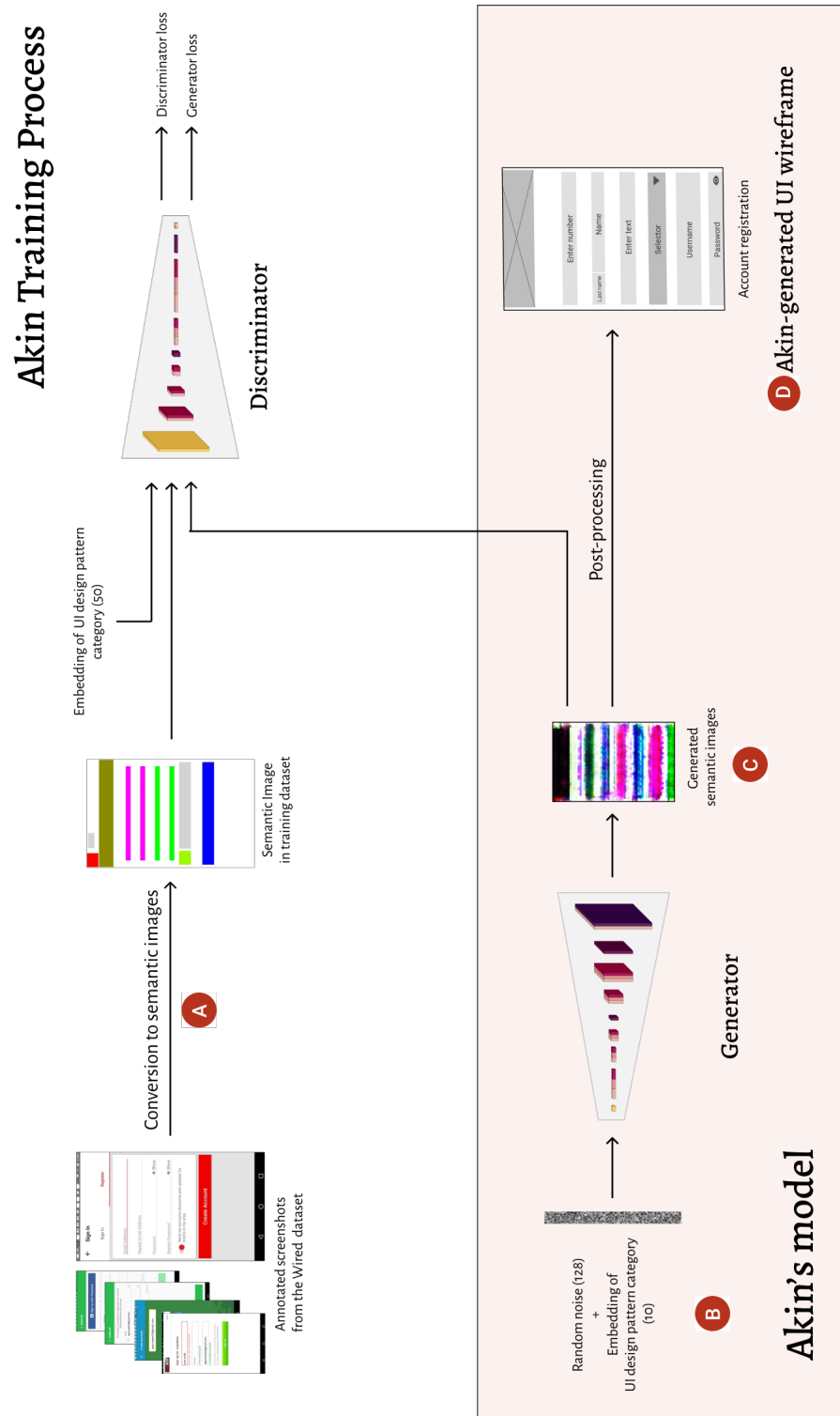
(a) Architecture of Akin's Generator model



(b) Architecture of Akin's Discriminator model

**Figure 12.1:** Architecture of Akin's conditional SAGAN model

followed by a Leaky ReLU activation layer. Similar to  $\mathcal{G}$ , after the third and fifth block, we add a Self-Attention layer and finally a Spectral 2D convolutional output layer. The output layer is flattened to obtain a boolean discriminator output, predicting whether the input semantic image is machine-generated or designer-made.



**Figure 12.2:** Akin is a UI wireframe generator trained with the semantic representations of UI images of 5 different UI design patterns from the Wired dataset. Akin's trained generator model (in the pale red box) generates different UI wireframes for a chosen UI design pattern.



## 12.4 CONFIGURATION & TRAINING PROCESS

With further empirical experiments, we finalized the hyperparameters for training Akin's SAGAN model as described below.

- **Image rescaling:** Rescaled to a square with 128 pixels on each side
- **Optimizer:** Adam optimizer with  $\beta_1 = 0.0$  and  $\beta_2 = 0.9$
- **Generator Learning rate:**  $1e^{-3}$
- **Discriminator Learning rate:**  $4e^{-3}$
- **Gradient Penalty ( $\lambda$ ) :** 10.0

For training the conditional SAGAN model, we set the training batch size to 16 to fit the available GPU memory. We neither preprocess nor augment the semantic images for training. We set up the training for 48 hours and stopped training at 7,000 epochs when the generated semantic images stabilized. For transparency and replicability of this research, we have open-sourced the codebase<sup>1</sup>.

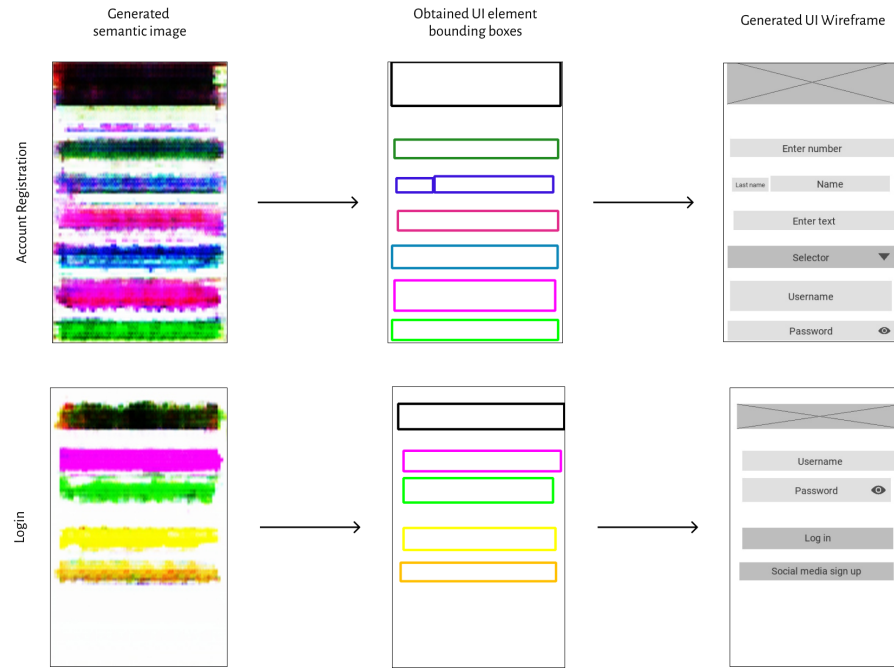
## 12.5 POST PROCESSING STEPS

As our SAGAN model's output is a semantic image (Figure 12.2, C), we post-process it to convert it to a UI wireframe (Figure 12.2, D). To achieve this, we created a script using OpenCV library created by Bradski (2000) in Python.

As a first step, this script upsamples the generated semantic image from 128x128x3 to 640x360x3. Then, using OpenCV algorithms, this script separates the generated semantic image into individual colour channels, applies an unsharp mask by subtracting the image with a Gaussian blur, thresholds the image using zeros, and finally merges all the channels into a single grayscale image. Further, the script runs OpenCV's contour detection algorithm on the obtained grayscale image to detect and extract all possible bounding boxes. By computing the mode of colour within these bounding boxes, the script also identifies the category of UI element. Thus, the script converts a generated semantic image to a list of UI element categories with their corresponding bounding box location and dimension. Finally, we use the semantic image annotation from the previous step by

*To convert the generated semantic image to UI wireframe, we created a post-processing script*

<sup>1</sup> <https://github.com/vinothpandian/akin-generator>



**Figure 12.3:** Conversion process of sample Akin generated semantic images to UI wireframes

stitching the appropriate UI element at their bounding boxes to generate a UI wireframe as shown in [Figure 12.3](#).

With this training process, we obtained the trained generator model (9) that can be given a noise vector of size 128 with a UI design pattern category to obtain multiple UI wireframes. This model (9) is our UI wireframe generator: Akin. Further to evaluate Akin's generative model, we used the Inception Score (IS) proposed by Salimans et al. (2016) to measure the quality of machine-generated wireframes and Fréchet Inception Distance (FID) proposed by Heusel et al. (2018) to measure the similarity between machine-generated and designer-made wireframes. We discuss the evaluation process in the next chapter.

---

## EVALUATION

---

To measure the quality of UI wireframes generated by Akin, we first evaluated it using quantitative metrics commonly used in GAN research. This chapter introduces the evaluation metrics we used, describes the evaluation methodology we followed, and the discusses evaluation results of the Akin: UI wireframe generator.

### 13.1 METRICS

Since the advancements in deep generative models, researchers have proposed several ways to measure the quality of the generated data appropriate for different domains. As Akin operates on the image domain by generated UI wireframes, we chose Inception Score (IS) and Fréchet Inception Distance (FID) to evaluate Akin's generator model after a literature review. In this section, we introduce both these metrics.

#### 13.1.1 Inception Score

Inception Score (IS) proposed by Salimans et al. (2016) is one of the most widely used metrics for evaluating the GAN models. It measures the diversity and quality of the generated samples. To measure IS, we require a pre-trained classification neural network, such as the Inception network proposed by Szegedy, W. Liu, et al. (2014), which acts as a feature extractor. This feature extractor must be trained to classify the images in the dataset, here UI wireframes from Wired dataset, to capture the distinct features of different classes and predict the probability of a given sample belonging to a particular class. Using this feature extractor, we can calculate the inception score by measuring the mean KL divergence between the conditional label distribution  $p(y|x)$  and the marginal distribution  $p(y)$  obtained from the dataset used to train the feature extractor. Salimans

*Inception Score measures the quality of machine-generated wireframes. A higher score is preferred*

et al. (2016) claims that IS shows a reasonable correlation with the quality and diversity of generated images. However, one major limitation of IS is that it fails to detect whether a model is agnostic to mode collapse, where the model can generate only one image per class.

### 13.1.2 Fréchet Inception Distance

*Fréchet Inception Distance (FID) to measure the similarity between machine-generated and designer-made wireframes. A lower score is preferred*

Fréchet Inception Distance (FID), introduced by Heusel et al. (2018), quantifies the quality of generated samples by comparing it with the real data. Similar to IS, FID uses a pre-trained classification neural network to measure the quality of generated samples. It uses the output of the penultimate layer in the Inception model to summarise the given data (either real or fake) into a continuous multivariate Gaussian. The FID score is the Wasserstein-2 distance—the difference between the real and fake Gaussians. Heusel et al. (2018) asserts that FID is more robust than IS and also consistent with human judgements. Another advantage of FID over IS is that FID can detect mode collapse; therefore, a model that generates only a single image for a given class could have a high IS score, but FID captures this failure, and its score worsens for less diverse generators.

## 13.2 METHODOLOGY

Both the evaluation metrics mentioned above requires a trained classification DNN for evaluating Akin's generator model. Therefore, we trained an Inception v3 model to classify the UI wireframes into 5 UI design patterns: Splash Screen, Login, Account Registration, Product Catalog, and Product Page.

*We fine-tuned the Inception Model with UI wireframes from the Wired dataset for evaluating Akin's IS and FID*

We modified the Inception model from Tensorflow 2.0 library for the classifier. As the existing model from the Tensorflow model zoo is pre-trained with the ImageNet dataset, we reinitialised the weights and trained the classifier from scratch only with UI wireframes. The output of our UI generator is a UI wireframe after applying the post-processing steps. To make the classifier effective on these UI wireframes, we applied the same post-processing steps by taking UI layout annotations from the Wired dataset and converting them to UI wireframes similar to generated UI wireframes. With this step, we obtained a dataset of 2,751 UI wireframes of 5 UI design patterns. Further, we split this dataset into 90% for training and 10% for evaluation.

We removed the existing model head from the Inception model and added a softmax layer with five outputs to enable the model to classify UI wireframes into 5 UI design patterns. We used an RMSprop optimiser with an initial learning rate of  $1e^{-3}$ , staircase exponential learning rate decay, and gradient clip of 2.0. To measure the training loss, we used a sparse categorical loss function. Finally, we trained this model with 90% of the 2,751 UI wireframes for 200 epochs. As the UI wireframes of 5 classes were not distributed equally, we also added class weights to tackle this class imbalance. This Inception model provides an accuracy of 97% on the 10% UI wireframes evaluation dataset, which was not used during the training. We used this Inception model to measure the IS and FID scores.

### 13.2.1 Results & Discussion

Using Akin's generator, we generated a batch of 100 UI wireframes with 20 UI wireframe for each UI design pattern. Then, we used the trained Inception model to classify these generated images into 5 UI design patterns and obtained the prediction probability from this model. Finally, using this information from the Inception model, we applied the respective formulae to obtain IS and FID scores.

Akin's generative model provides

- **Inception Score:** 1.63 (SD=0.34)
- **Fréchet Inception Distance:** 297.19

In general, a higher IS score is preferred, whereas a lower FID is preferred. However, as the existing research on UI layout generation does not provide a baseline metric, we could not compare the IS and FID scores with other UI layout generation models. Hence, to further analyse the quality of UI wireframes generated by Akin and understand the designer's satisfaction while using Akin, we conducted three quantitative user evaluations and a qualitative semi-structured interview; these are described in the next chapter.



---

## USER EVALUATION

---

In the previous chapter, we evaluated Akin's generator model with commonly used GAN quantitative metrics. Further, we conducted three quantitative and qualitative studies to understand Akin generated UI wireframe quality and designer satisfaction in using Akin before starting their LoFi design process. This chapter explains these studies successively.

### 14.1 USER EVALUATION OF UI WIREFRAMES

To understand the quality of the wireframe from a designer's perspective, we aimed to evaluate Akin generated UI wireframes with UI/UX designers. Upon reviewing the literature review, we identified two appropriate standard quantitative studies to measure the quality of generated UI wireframes: rating-preference judgment and rapid scene categorization. This section discusses these studies briefly.

*We conducted two standard quantitative studies to measure the quality of generated UI wireframes*

#### 14.1.1 Participants

We used purposive and snowball sampling to recruit 15 participants for these studies: 7 UI/UX designers and 8 HCI grad students (9 male and 6 female). These participants were between 23 to 35 years of age and had 1 - 5 years of prior prototyping experience (Mean = 1.2, SD = 1.27).

#### 14.1.2 Study Design Decisions

Prior research by Tractinsky et al. (2006) and Tuch et al. (2012) shows that humans can report characteristics of a design at a short glance in the range of 50 milliseconds (ms) to 500 ms. Tuch et al. (2012) noted that even at a short glance of 17 ms, users could make snap judgements of the design. Further, Tractinsky et al. (2006) noted that the users rating were

correlated for 50 ms and 500 ms. Therefore, we chose a rapid scene categorization study. Further, we aimed to measure the quality of Akin generated UI wireframes and compare them with designer-made UI wireframes; hence, we also decided to conduct a rating-preference judgement study. We combined these investigations into two phases of a user study.

## 14.2 STUDY DESIGN & MEASUREMENTS

*In rapid scene categorization, we aim to understand whether designers, in a snap judgement, perceive a given UI wireframe was Akin-generated or Designer-made*

In the first phase of the study, we started with rapid scene categorization. In this study, we flashed Akin-generated and designer-made wireframes in a random order to the participants for 500 ms, 1,500 ms, 2,750 ms, 4,000 ms, and 5,000 ms intervals. The independent variables in this study are the UI wireframe image and the number of milliseconds the UI wireframe image is shown to the participant. The dependent variable is the designer's choice of whether the shown UI wireframe is designer-made or Akin generated.

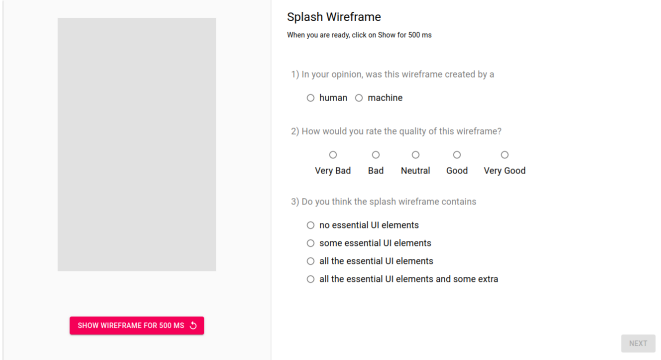
*In rating-preference judgment study, we aim to understand designer's subjective rating of Akin-generated wireframes and whether it contains all necessary UI elements*

The second phase of the study uses the same UI wireframe but shown indefinitely to conduct a rating-preference judgment study. In this study, we asked the participants to provide their subjective rating regarding the quality of the wireframe using a Likert scale of 1-5 and whether they think the wireframe contains all essential UI wireframes for the given UI design pattern. The independent variables in this study are the UI wireframe image and its UI design pattern type. The dependent variables are the subjective rating of the given UI wireframe and their opinion of whether the UI wireframe contains all essential UI elements.

Each participant was shown 50 UI wireframes: 25 designer-made and 25 Akin-generated. For each UI wireframe, we asked the following questions.

1. In your opinion, was this wireframe created by a human or a machine?
2. How would you rate the quality of this wireframe on a scale of 1-5?
3. Do you think the login wireframe contains?
  - no essential UI elements
  - some essential UI elements
  - all the essential UI elements
  - all the essential UI elements and some extra





**Splash Wireframe**  
When you are ready, click on Show for 500 ms

1) In your opinion, was this wireframe created by a  
☐ human ☐ machine

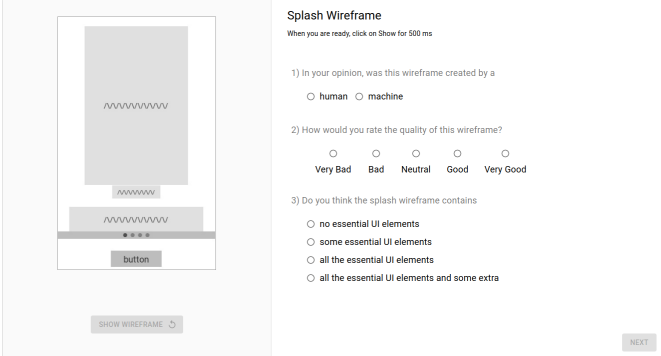
2) How would you rate the quality of this wireframe?  
☐ Very Bad ☐ Bad ☐ Neutral ☐ Good ☐ Very Good

3) Do you think the splash wireframe contains  
☐ no essential UI elements  
☐ some essential UI elements  
☐ all the essential UI elements  
☐ all the essential UI elements and some extra

SHOW WIREFRAME FOR 500 MS

NEXT

(a) Before the participant clicks "Show Wireframe for 500 ms" button, only a UI wireframe placeholder is shown



**Splash Wireframe**  
When you are ready, click on Show for 500 ms

1) In your opinion, was this wireframe created by a  
☐ human ☐ machine

2) How would you rate the quality of this wireframe?  
☐ Very Bad ☐ Bad ☐ Neutral ☐ Good ☐ Very Good

3) Do you think the splash wireframe contains  
☐ no essential UI elements  
☐ some essential UI elements  
☐ all the essential UI elements  
☐ all the essential UI elements and some extra

button

SHOW WIREFRAME

NEXT

(b) After the participant clicks "Show Wireframe for 500 ms" button, the UI wireframe is flashed for 500 milliseconds and after participant answers question 1, they can again click "Show Wireframe" button to show the wireframe indefinitely

**Figure 14.1:** Screenshots of the web application to conduct rapid scene categorization and rating-preference judgment studies before and after the participant clicks "Show Wireframe for 500 ms" button

### 14.2.1 Apparatus

To conduct these studies, we uniformly sampled 25 designer-made images used in the training dataset of the Inception model, and 25 Akin generated UI wireframes from the evaluation study described in the previous chapter. Each UI design pattern was represented in these 50 UI wireframes equally: 5 designer-made UI wireframes and 5 Akin generated UI wireframes for each UI design pattern.

To conduct the studies described in the previous subsection, we created a web application<sup>1</sup>. Each screen in the web application is divided into two sides. The left side of the web application contains a UI wireframe placeholder with a button that will show a UI wireframe for the mentioned

*We created a web application questionnaire to conduct the study remotely*

<sup>1</sup> <https://uisketch-survey.web.app/>

milliseconds on the placeholder when clicked. On the right side of the web application, we display the UI wireframe type and a form with three questions described in the previous subsection.

Figure 14.1 shows the questionnaire before and after the "Show Wireframe from 500 ms" button is clicked. Only a UI wireframe placeholder is shown before the participant clicks the "Show Wireframe for 500 ms" button. After the participant clicks the "Show Wireframe for 500 ms" button, the UI wireframe is flashed for 500 ms, and after the participant answers the first question, they can again click the "Show Wireframe" button to show the wireframe indefinitely.

#### 14.2.2 Procedure

Due to the ongoing corona crisis, this study was conducted remotely via video conferencing. Before the study, we explained the purpose of the study and requested the participants to provide informed consent. Once the participant agreed, we provided them with the web application link and asked them to share their screen during the course of the study.

Before the study started, we showed them a sample screen to get comfortable with the study setup of clicking on "Show wireframe for given ms" and then "Show wireframe" to show it indefinitely. Once the participant agreed to proceed further with the study, the web application created a secure key for authentication and stored it in their browser's internal database. This step helps restore the web application from the point the participant left off in case of internet disconnection.

During the study, the participants were shown all 50 UI wireframes in random order. For each UI wireframe, the participant was shown a screen with only the UI wireframe placeholder and a button that allowed them to show the wireframe for one of 500 ms, 1,500 ms, 2,750 ms, 4,000 ms, and 5,000 ms intervals. Once the participants clicked this button, the UI wireframe is flashed for the specified milliseconds. Then the participant was asked to answer whether the flashed UI wireframe was designer-made or Akin generated. Once the participant answered this question, we asked them to click on the "Show wireframe" button to display it indefinitely. Here, the participant was allowed to examine the UI wireframe for an indefinite time and asked to rate the quality of the show UI wireframe and their opinion on whether it contains all essential UI elements. During this phase, we did not allow the participant to modify their answer to the first question. Likewise, we asked the participants to proceed with all 50 UI

wireframes. On average, participants took around 1 hour to complete the study.

### 14.2.3 Results & Discussion

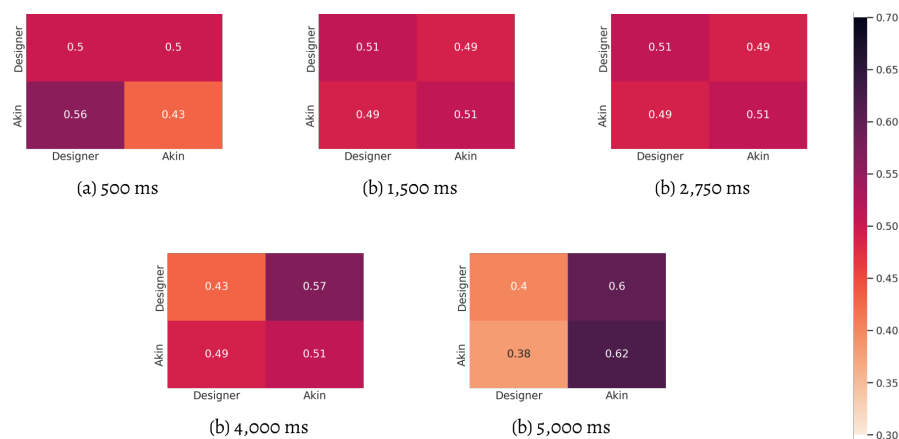
We analyzed the results of both rapid scene categorization and rating-preference judgment studies to understand the quality of Akin generated wireframes and compare them with designer made UI wireframes. In the upcoming sections, we report the results of each study and discuss our inference from the results.

#### 14.2.3.1 Rapid scene categorization

From the study results, we observe that for intervals less than 2,750 ms, the participants identified Akin-generated wireframes as designer-made ~50% of the time, similar to a fair coin toss probability. This result shows that the designers could not distinguish Akin-generated wireframes from designer-made wireframes on a quick look. Figure 14.2 (a, b, c) visualizes this result using a confusion matrix.

For 4,000 ms and 5,000 ms intervals (Figure 14.2 d, e), we observe that the participants could distinguish Akin-generated UI wireframes from designer-made wireframes to a certain extent. Figure 14.3 further establishes this result that, on snap-judgement, participants classify Akin-generated UI wireframes as designer-made but given time, they could distinguish them.

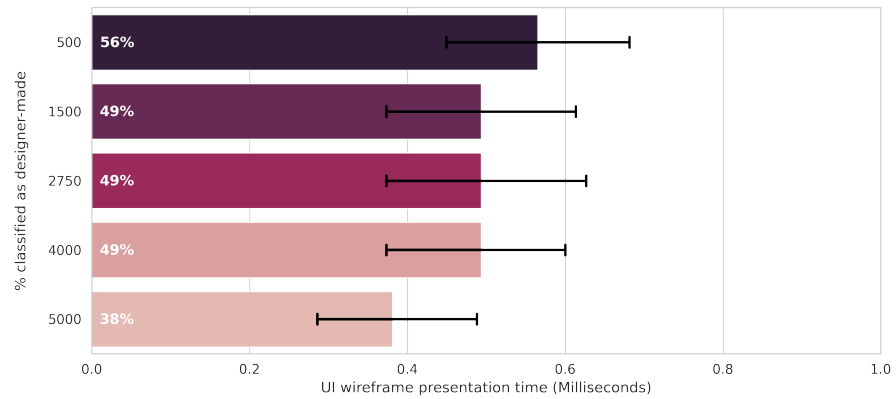
*Rapid scene categorization result shows that the designers could not distinguish Akin-generated wireframes from designer-made wireframes on a quick look*



**Figure 14.2:** Confusion matrix for rapid scene categorization study comparing participants' accuracy of identifying a given UI wireframe as Akin-generated or designer-made

**Table 14.1:** Results of rapid scene categorization study across all five intervals

Created by	Identified as	Time Interval (ms)				
		500	1,500	2,750	4,000	5,000
Designer	Designer	0.50	0.51	0.51	0.43	0.40
	Akin	0.50	0.49	0.49	0.57	0.60
Akin	Designer	0.56	0.49	0.49	0.49	0.38
	Akin	0.43	0.51	0.51	0.51	0.62

**Figure 14.3:** Percentage of Akin-generated UI wireframes classified by the participants as designer-made for the given UI wireframe presentation time intervals

Surprisingly, we can also notice that, when given 5,000 ms, the participants also identify designer-made UI wireframes as Akin-generated wireframes. To understand this further, we discussed the results with participants later. From this discussion, we understood that participants identified Akin-generated wireframes only when their alignment was slightly inexact than designer-made wireframes. When given more time, the participants could not make a confident decision, and therefore some assumed designer-made wireframes as Akin-generated.

Overall, this study shows that the participants could not confidently distinguish Akin-generated wireframes from designer-made wireframes. [Table 14.1](#) summarizes the results of this study across all time intervals.

#### 14.2.3.2 Rating-preference judgement

From the results of this study, summarized in [Table 14.2](#), we conclude that the rating of Akin-generated wireframes (Mean = 2.98, SD = 0.08) is approximately equal to the designer-made wireframes (Mean = 2.90, SD = 0.11) for all UI design patterns.

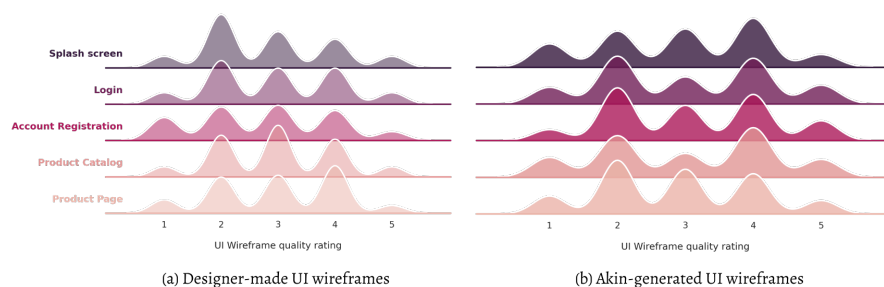
**Table 14.2:** Results of rating-preference judgement study for all UI design pattern types

UI design pattern	Akin-generated	Designer-made
Splash screen	2.95	2.82
Login	3.01	2.95
Account Registration	3.06	2.75
Product Catalog	3.02	2.97
Product Page	2.86	3.00
<b>Mean</b>	2.98	2.90
<b>SD</b>	0.08	0.11

In general, for Splash, Login, and Account creation screens, the participants prefer Akin-generated UI wireframes over designer-made wireframes. However, for the Product Catalog and Product Page, the participants' preference was much closer.

From [Figure 14.4](#), we can see that the distribution of the subjective participant rating for different UI design patterns of designer-made UI wireframes is similar to the distribution of ratings of Akin-generated UI wireframes.

*Rating preference judgment results show that users rate the Akin-generated UI wireframes as good as designer-made wireframes*

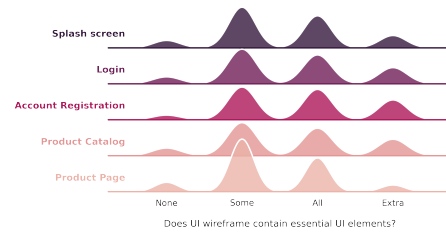
**Figure 14.4:** Distribution of participants' subject rating of Akin-generated and designer-made UI wireframes for all UI design pattern types

In addition to the subjective rating, we also asked the participants to express their opinion on whether the UI wireframe shown contain all essential UI elements. From [Table 14.3](#) Approximately 76% of the participants consider that Akin-generated UI wireframes contain some to all necessary UI elements. Only ~16% of the participants believe that the Akin-generated UI wireframes contain no essential UI elements. [Table 14.5](#) visualizes the distribution of participants and their choices during this study for each UI design pattern type.

These two studies confirm that the quality of UI wireframes generated by Akin is adequate for usage in the further prototyping process. In ad-

UI Design Pattern Type	% of Participants			
	None	Some	All	Extra
Splash screen	8.00	<b>44.00</b>	34.67	13.33
Login	8.00	<b>40.00</b>	33.33	18.67
Account Registration	5.33	<b>37.33</b>	34.67	22.67
Product Catalog	9.09	<b>38.96</b>	32.47	19.48
Product Page	6.58	<b>51.32</b>	32.89	9.21
<b>Total</b>	7.94	<b>42.33</b>	33.60	16.14

**Table 14.3:** Percentage of participants who considered whether Akin-generated UI wireframes of given UI design pattern type contains all essential UI types



**Figure 14.5:** Distribution of participants who considered whether Akin-generated UI wireframes of given UI design pattern type contains all essential UI types

dition to this study, we aim to measure user satisfaction using Akin to generate UI wireframes before a designer starts their LoFi prototyping process. Therefore, we conducted a user satisfaction study using the After Scenario Questionnaire (ASQ). We discuss this study and its results in the upcoming section.

### 14.3 USER SATISFACTION STUDY

From the rapid scene categorization and rating-preference study, we understood that Akin generates UI wireframes with acceptable quality. To further understand designer satisfaction while using Akin, we conducted a quantitative and qualitative study. To conduct this study, we deployed the Akin UI wireframe generator as a web API and created an Adobe XD plugin to display the generated UI wireframes. This section explains the study setup, participants, methodology, and result.

#### 14.3.1 Web API & Adobe XD plugin

*We created a web API for Akin to create a modular and reusable interface*

A key focus of Akin is to provide a modular and reusable interface to generate UI wireframes and support designers before the LoFi prototyping process. To achieve this, we created a Web API to providing a user-friendly abstraction and access to Akin's generator model. We wrote this API in Python using PyTorch and FastAPI frameworks.

This Web API<sup>2</sup> accepts a UI Design pattern type in the HTTP request. This pattern type can be one of the following five values: *login*, *account\_creation*, *product\_listing*, *product\_description*, and *splash*. The API triggers Akin's gen-

<sup>2</sup> <https://akin.blackbox-toolkit.com/>

erator model, generates six UI wireframes and responds with a list of UI element categories and their location, dimension in JSON format. We have limited to six UI wireframe generation due to the server capacity.

We utilized this Web API and implemented a plugin for Adobe XD<sup>3</sup>. Adobe XD is one of the popular free-to-use UI prototyping tools. We created this plugin using Typescript. This plugin accepts a UI design pattern as input and calls the Web API to generate UI wireframes for the given UI design pattern input. Then, it renders the wireframes on the prototyping tool, which can further be edited and reused in Adobe XD by the UI designer. Figure 14.6 shows a screenshot of the Akin plugin and its results in the Adobe XD prototyping tool.

#### 14.4 METHODOLOGY

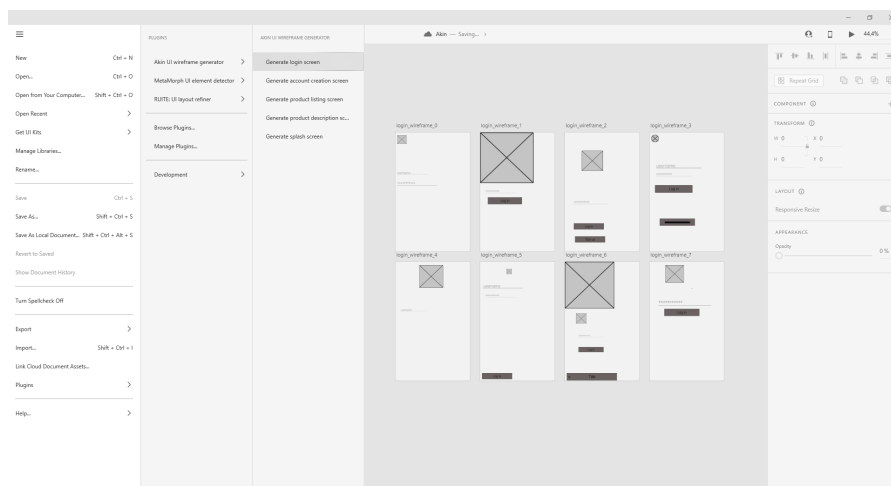
To assess the designers' satisfaction in utilizing AI before their traditional prototyping process by generating UI wireframes, we conducted a quantitative study using After-Scenario Questionnaire (ASQ), created by Lewis (1991b), followed by semi-structured interviews (15 min) to obtain qualitative feedback. The ASQ measures user satisfaction on the following three scales.

*We used the standard After-Scenario Questionnaire (ASQ) for measuring user satisfaction in using Akin*

Q1 Ease of completing the tasks

Q2 The amount of time it took to complete the tasks

3 <https://www.adobe.com/products/xd.html>



**Figure 14.6:** Screenshot of Akin plugin and the generated wireframes in Adobe XD prototyping tool

Q3 The supporting information provided while completing the tasks

#### 14.4.1 *Participants*

We used purposive and snowball sampling to recruit 10 UI/UX designers (5 male and 5 female). The participants were between 24 to 32 years of age and had 1 - 6 years (Mean = 2.2, SD = 1.86) of prior prototyping experience.

#### 14.4.2 *Scenario & tasks*

The participants were given a scenario to interact with the Akin plugin implemented in Adobe XD to generate LoFi prototypes and then use them to create three screens of an e-commerce application. They were instructed to complete the following three tasks.

1. Use Akin plugin to generate 3 UI screens (e.g. Login, Product Page, and Product Description).
2. Choose preferred UI screens and correct them if necessary
3. Connect the chosen screens into a UI prototype

We asked participants to generate as many UI wireframes as preferred until they are satisfied during this study. Task 1 was randomized with a different set of screens for different participants. Also, keeping the flow of prototyping fidelity transformation in mind, we did not limit designers adding or removing UI elements to the generated screens.

#### 14.4.3 *Material & Apparatus*

Participants performed the aforementioned tasks remotely. We connected the participants via video conferencing and provided them access to our computer, which already contains Adobe XD with Akin plugin installed.

#### 14.4.4 *Procedure*

*Study was conducted  
remotely with screen  
sharing*

Due to the ongoing COVID-19 pandemic, we conducted this study remotely via video conferencing. Once the participant agreed to participate in the study, we acquired their informed consent and then invited them to



a virtual meeting. Before the study, we briefed the participants about the study's purpose and the assigned task. Then, participants were given time to explore various features of the Akin plugin and Adobe XD features to familiarize themselves with the environment and test the remote control setup.

After this brief exploration period, participants performed the assigned tasks and provided feedback using a think-aloud protocol. Once the participant completed the tasks, they were asked to fill the ASQ. On average, participants took around 15 minutes to complete the tasks and fill the questionnaire. After the study, we conducted semi-structured interviews (~15 mins) to understand the participant's view on the AI assistance before LoFi prototyping approach and better understand and interpret the ASQ results.

#### 14.4.5 Results & Discussion

Akin received an above-average satisfaction level for all three questions in the ASQ: ease of completing the tasks (4.6), the amount of time it took to complete the tasks (5.7), and the supporting information provided when

**Table 14.4:** Results of the ASQ study with participant's prior experience in UI prototyping

ID	Prototyping experience (in years)	Q1	Q2	Q3
P1	4	5	7	5
P2	2	3	6	4
P3	1	5	6	6
P4	2	5	5	4
P5	1	4	7	5
P6	1	5	5	6
P7	4	4	5	4
P8	1	4	6	5
P9	6	5	4	4
P10	1	6	6	7
<b>Mean</b>		4.6	5.7	5.0
<b>SD</b>		0.84	0.95	1.05

*Akin received an above-average satisfaction level for all three questions in the ASQ*

*Novice users were more satisfied using Akin than experienced users*

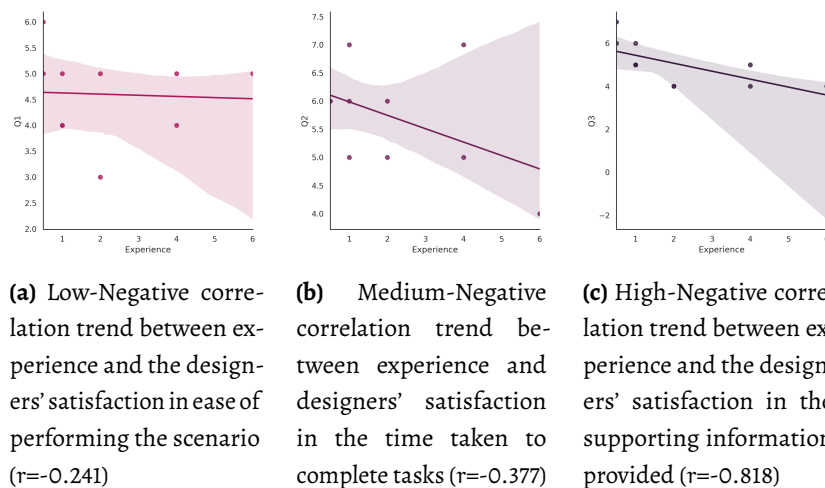
*Both novice and experienced UI designers found AI assistance before LoFi prototyping to be exciting and useful*

completing the tasks (5.0). Table 14.4 summarizes the ASQ score from each participant along with their UI prototyping experience.

We further analyzed the correlation between the experience of the UI designer and their ASQ satisfaction score. Unlike the ordinal values of the 7-point Likert scale in ASQ, the prototyping experience of a UI designer is on the ratio scale. Therefore, we converted the experience to an ordinal scale using average ranking and used rank correlation coefficient by Spearman (1904) to measure the correlation between them.

Overall, these results show a negative correlation between the years of experience and their satisfaction in using Akin as a prototyping assistant. Specifically, there is a low-negative correlation between the experience of the UI designer and their satisfaction in ease of performing the scenario ( $r_{\text{experience} \times Q1} = -0.241$ ) and a medium-negative correlation between experience and the designers' satisfaction in the time taken to complete tasks in the scenario ( $r_{\text{experience} \times Q2} = 0.405$ ). Notably, a high-negative correlation was found ( $r_{\text{experience} \times Q3} = 0.107$ ) between experience and the designers' satisfaction with the supporting information provided. Figure 14.7 visualizes the correlation trend between precision and the three questions from ASQ. To understand this trend, we further analyzed our semi-structured interview results.

From the qualitative feedback provided by our participants, both novice and experienced UI designers found this approach of adding AI assistance in generating multiple wireframes of UI design patterns to be helpful during their traditional UI design process. Furthermore, novice designers were excited to use such an approach for all possible use cases, whereas



**Figure 14.7:** Spearman rank correlation between UI element detection precision and the three questions from ASQ

experience designers mentioned that they would use it when they were in a hurry to provide quick results.

Experienced designers found the system to be slower (Figure 14.7b) than they expected as it currently generates only 6 UI wireframes at one button click. However, this is a limit imposed due to system limitations during testing; generally, Akin can generate any given number of UI wireframes. We expect to investigate whether removing this limit improves their satisfaction in our future work.

Also, the experienced designers found the supporting information provided could be improved (Figure 14.7c, c) by allowing designers to add constraints and partial UI wireframe generation.



*I would definitely use it because it would reduce the cost of development and increase the speed.*

-P4



*Wireframes are neat and consistent in design, but they are not aligned properly. That adds some extra effort*

-P7



*Wireframes are good for simpler patterns but not for product catalog. It would be nice if I can ask it (Akin) to generate only the part below App bar*

-P10

Using AI as an assistant before LoFi prototyping using Akin was generally regarded as an exciting and practical approach based on this qualitative feedback. All participants were interested in adopting such an approach provided the UI wireframes were aligned during generation. In the upcoming chapter, we summarize our findings while creating and evaluating Akin, describe the identified limitations and future work.



---

## SUMMARY & FUTURE WORK

---

In this part of the thesis, we introduced Akin, a UI wireframe generator. Akin assists UI designers before the LoFi prototyping process by allowing them to choose a UI design pattern and provides them with multiple UI wireframes for the chosen UI design pattern. It uses a fine-tuned Self-Attention Generative Adversarial Network (SAGAN) trained with 500 UI wireframes of 5 android UI design patterns. Upon evaluation, Akin's generative model provides an Inception Score of 1.63 ( $SD = 0.34$ ) and Fréchet Inception Distance of 297.19. We further conducted user studies with 15 UI/UX designers to evaluate the quality of Akin-generated UI wireframes. The results show the participants consider Akin-generated UI wireframes ( $Mean = 2.98, SD = 0.08$ ) as good as designer-made wireframes ( $Mean = 2.90, SD = 0.11$ ). Also, the UI designers could not distinguish Akin-generated UI wireframes from designer-made wireframes (50% accuracy).

Additionally, we conducted a quantitative study using After-Scenario Questionnaire (ASQ) followed by semi-structured interviews to understand the designer satisfaction using Akin. The results indicated that designers experience an above-average satisfaction level towards ease of task completion (4.6), time taken (5.7), and supporting information (5.0) upon utilizing AI assistance for generating multiple UI wireframes before they start their prototyping process. Their qualitative feedback indicated that both novice designers and experience designers found AI assistance helpful in addition to the traditional UI prototyping process.

This research addresses a part of RQ 2, 3 and 4 of this thesis. Also, it fulfils the two significant research gaps we identified. Unlike the prior research, we distinguish UI wireframes into different UI design patterns and trained a conditional GAN to learn the underlying representative UI layouts of a given UI design pattern. Moreover, we evaluated Akin using standard quantitative machine evaluation and with users to understand

the quality of generated wireframes and the satisfaction of designers in using AI assistance before the LoFi prototyping process.

Further, through the qualitative study, we identified a few limitations in Akin, which we list below. We aim to improve Akin by addressing these limitations in our future work.

One notable issue that many designers mentioned during the user study is that Akin generated UI layouts are not aligned pixel-perfect. As a solution, we plan to couple Akin with an alignment engine, such as RUIITE, which is explained in the upcoming part, to improve the alignment quality of the generated UI wireframes. Another feature we would like to add is to specify user constraints and generate a partial UI layout.

In the future, we plan to add a frontend to Akin, which allows the designer to enter styling information such as colour palette, typography, and a UI design pattern to generate customized UI screens. Also, as the current system only detects five UI design patterns, we would like to add more UI design pattern categories, increase the dataset size, and retrain Akin.

## PART IV

# RUIITE

In this part, we introduce RUIITE, a UI wireframe refiner that assists UI designers *during LoFi prototyping*. RUIITE aligns and groups UI elements in a given UI wireframe; thus, enabling designers to prototype quickly. We created RUIITE using a Transformer-Encoder model and training it using 35,072 UI annotations from the SynZ dataset. RUIITE's model provides satisfactory results on almost all evaluation metrics: alignment score (0.87), improvement margin (~38%), grouping accuracy (20%), and mean Average Precision (58.53%). Our user evaluation with the ASQ to assess user satisfaction indicates that designers experience an above-average satisfaction level towards ease of task completion (5.3), time taken (6.6), and supporting information (6.2) upon utilising RUIITE during the LoFi prototyping process. The qualitative feedback indicated that both novice and experienced participants preferred refinement of UI wireframe using AI in a button-click in addition to the traditional UI prototyping process.



<https://ruiite.blackbox-toolkit.com>





---

## PROPOSED SOLUTION

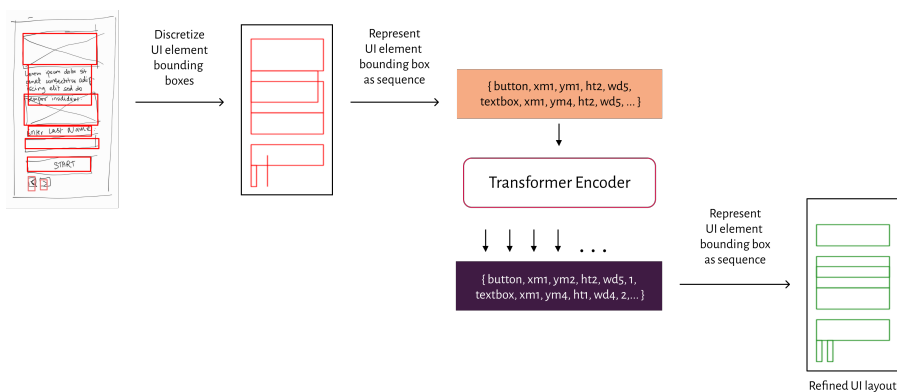
---

RUIITE is a UI layout refiner that optimises UI wireframe layouts using a Transformer Encoder model, thereby assisting UI designers during the LoFi prototyping process by aligning and grouping UI wireframes. This research expands from the prior research in automatic generation and refinement of document or magazine layouts and introduces a novel Transformer-Encoder DNN for UI layout refinement. This part of the dissertation is published at IUI 2021 (Rahman et al., 2021) created under my supervision based on my research.

*RUIITE assists designers **during** their LoFi prototyping task by aligning and grouping UI wireframe.*

### 16.1 BACKGROUND

Prior research in the automatic creation of document or magazine layouts explored assisting designers during their design process by providing them with layout suggestions. These research projects mostly use classic



**Figure 16.1:** RUIITE improves the UI layout alignment by optimizing the location and dimension of UI elements on a screen. It takes a list of UI elements and their respective bounding boxes as input and optimizes the layout to be aesthetically appealing using a Transformer Encoder

*Most of the prior research on UI layout refinement was conducted with classical pattern recognition or algorithmic models.*

pattern recognition algorithms for inferring and suggesting alternate layouts for documents or magazines.

Similarly to refining UI layouts, in the last decade, few research projects attempted to apply classical pattern recognition. Few such research projects by O'Donovan et al. (2014) endeavoured to measure layout aesthetics and refine them by computing and minimising energy functions. In their first publication, O'Donovan et al. (2014) considered features of graphical layouts, such as white space, symmetry, and alignment, and devised an energy function of these features. Then, by minimising this energy function using simulated annealing, they create a refined layout. However, according to the authors, this approach is not scalable and takes up to 40 minutes to generate an optimised layout. In their follow-up publication, O'Donovan et al. (2015) expanded their energy function model by considering additional aesthetic features and provided alternate design suggestions. Further, they allowed users to add a constraint to the energy function, thus customising layouts' refinement. Another recent research, GRIDS by Dayama et al. (2020) uses Mixed Integer Linear Programming for refining MeFi GUI layouts. This system operates on a mathematical optimisation of general grid layout principles.

*Only one recent research applied DNNs for UI wireframe refinement.*

Recently, Lee et al. (2020) created Neural Design Network to generate and refine UI wireframes. In this research, the authors focus on UI wireframe generation; as a side-effect to generate an aligned UI wireframe, they also explored UI layout refinement. The most exciting approach taken by the authors is to represent UI layouts as graphs. Such a layout graph is constructed by representing UI elements as nodes and their relationship (location above, below) as edges. The Neural Design Network is constructed using a Graph Convolution Network (GCN) that generates or refines UI wireframes as layout graphs. This GCN is trained using a subset of the RICO dataset and provides a Frèchet Inception Distance (FID) score of  $143.51 \pm 22.36$  upon evaluation. However, the authors did not evaluate the UI refinement standalone, and also, they did not assess the quality of the refined UI wireframes with designers.

## 16.2 IDENTIFIED GAPS

*Prior research were either deterministic, slow or have not been evaluated by designers to uncover their needs and desires*

In summary, most of the prior research in this domain, such as GRIDS by Dayama et al. (2020), utilises classical algorithmic techniques for optimising UI layouts: assuming a grid-based layout and applying a deterministic 2d grid packing algorithm. Recent research on this domain, such as the

Neural Design Network by Lee et al. (2020), concentrates on UI layout generation and, as a byproduct, refines the generated UI layout. The major shortcoming of the researches mentioned above is that they are either deterministic, slow or have not been evaluated by designers to uncover their needs and desires.

### 16.3 PROPOSED SOLUTION

As a solution to the identified research gaps, we propose RUIITE, a UI layout refiner. The goal of RUIITE is to assist designers during LoFi prototyping by refining and grouping UI wireframes. Also, we aim to create a model that learns layout styles from real-world UI and use it to refine any given UI wireframe.

RUIITE uses a Transformer Encoder model to align a misaligned UI layout. We define a UI layout of a UI screen as a list of UI elements present in that screen and their respective location and dimensions (bounding-box). To train and validate RUIITE, we used 39,456 UI layouts annotations from the SynZ dataset (Section 6.3.1). We added random noise to a subset of these UI layouts to simulate misaligned UI layout and provided this as training data. The model learns to align this misaligned layout to the ground-truth aligned layout. We then evaluated this trained model using the mean average precision (mAP) of coco detection metrics by T. Y. Lin et al. (2014) using different Intersection over Union (IoU) thresholds, Alignment Score proposed by Koch et al. (2016), grouping accuracy, and alignment improvement score.

Further, to quantitatively measure the user satisfaction of designers upon using RUIITE during LoFi prototyping, we used the standard After Scenario Questionnaire with 10 participants: 5 UI/UX designers and 5 HCI grad students. Finally, to further understand the designer satisfaction results, we followed the user satisfaction study by conducting a qualitative semi-structured interview.

In the upcoming chapter, we discuss the experiments we conducted to determine the data representation of UI wireframe layouts and the model architecture of RUIITE.

**Model:** Transformer Encoder

**Dataset:** 39,456 UI layout annotations from the SynZ dataset

**Evaluation:** Four AI metrics, one quantitative and qualitative user evaluation with UI/UX designers



---

MODEL ARCHITECTURES & DATA REPRESENTATIONS

---

Prior research projects on magazine and document layout refinement used traditional pattern recognition and algorithmic models. However, to create such models, we have to carefully select and understand the features of a layout. As an alternative approach, recent advancements in DNNs enables us to utilise supervised learning techniques and learn from existing features of a UI wireframe layout. We utilised these recent advancements to create RUIE: UI layout refiner. This chapter describes the experiments we conducted with different data representations of UI layouts and different model architectures to create RUIE. Detailed information of these experiments and results are published at Rahman et al. (2021) created under my supervision based on my research.

### 17.1 MODEL ARCHITECTURES

As recent research in applied AI in UI design did not emphasise UI layout alignment models, we took inspiration from the generative model research and applied them for UI layout refinement. We identified two promising DNN architecture for building a UI layout refiner through literature review: Graph Convolution Networks (GCNs) and Transformers. We describe them briefly in the upcoming sections.

*We experimented with two model architectures to create RUIE*

#### 17.1.1 Graph Convolution Network

Graphs are an abstract yet effective way of representing real-world data and their relationship with each other. A graph consists of a set of nodes (data) and a set of edges between them (relationships). They are commonly used to represent a network of roads on a map, the relationship between people in a social network, or links between computers on the internet.

*Graph Convolutional  
Networks can operate  
directly on graph data  
structures*

Once a real-world scenario is modelled as a graph, it can be effectively used to infer various statistical information about it.

Kipf et al. (2017) developed the Graph Convolution Networks (GCN), a DNN for semi-supervised learning on graph-structured data which can directly operate on graphs. GCN approximates a function of features on a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  by taking two inputs: features of nodes as a matrix  $\mathcal{X}$  with dimensions  $\mathcal{N} \times \mathcal{D}$  ( $\mathcal{N}$ : number of nodes,  $\mathcal{D}$ : number of input features). Using non-linear neural network layers, it learns the features from the inputs and produces the output as a matrix  $\mathcal{Z}$  with dimensions  $\mathcal{N} \times \mathcal{F}$  where  $\mathcal{F}$  is the number of output features per node.

Building on this research, many research projects, as listed by Zhou et al. (2020), have explored employing GCNs in various domains such as text, image, and structural domains. Related to UI design, Lee et al. (2020) utilised GCNs to create Neural Design Network for generating and refining UI layouts.

### 17.1.2 Transformers

Sequence-to-Sequence (Seq2Seq) networks are a subclass of DNNs that are trained to convert sequences from one domain (e.g. English sentences) to sequences in another domain (translated to German sentences). Several Seq2Seq networks, such as Recurrent Neural Networks (RNNs), Long-short-term memory (LSTMs) and Gated Recurrent Units (GRU), were proposed for accomplishing these sequence conversion scenarios. As a variant, Vaswani et al. (2017) introduced the Transformer architecture with Self-Attention to improve the Seq2Seq tasks.

*Transformers are  
sequence-to-sequence  
models that are  
currently popular in  
Natural Language  
Processing tasks*

A Transformer architecture consists of an encoder and a decoder where the encoder encodes the input sequence to an intermediate hidden state, and the decoder then decodes it into the target sequence. Transformer architectures have been popularly used in chatbots, text summary and machine translation tasks. Recently, Gupta et al. (2020) demonstrated using the Transformer architecture to generate UI layouts.

We utilised GCN and Transformer models for creating RUIE, a UI layout refiner. As discussed in this section, the input domain of GCN and Transformer was different. Therefore, we represented UI wireframes as graphs and as a sequence to experiment with these models. We describe them further in the following section.

## 17.2 DATA REPRESENTATIONS

RUIITE aims to refine the alignment and grouping of a given UI wireframe layout. Therefore, to address RQ2 and train the DNNs described above, we experimented with two different data representation of a UI wireframe layout: graphs and sequence. To train such DNNs, we require large-scale annotated UI layout information; hence, we chose to use annotation files from the SynZ dataset (Section 6.3.1) to train RUIITE. This section describes the two different data representations of these annotation files.

*We experimented with two data representation of UI wireframe layouts to train and evaluate RUIITE*

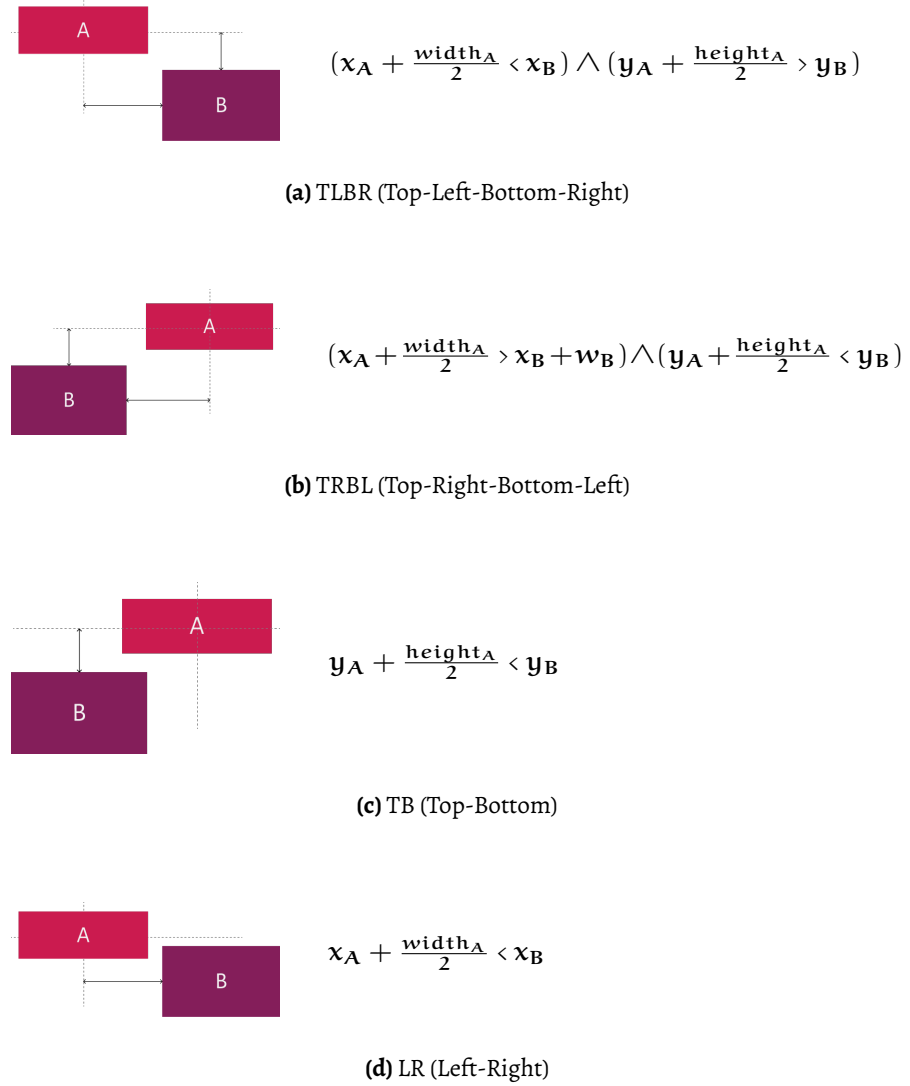
### 17.2.1 Graph

A UI wireframe annotation is represented as a list of constituent UI elements and their corresponding location and dimensions within the UI wireframe. These UI elements are aligned and grouped with respect to each other to provide an aesthetic and friendly interface. This alignment and grouping can be considered a relationship between the UI elements present in the UI wireframe. Lee et al. (2020) used this notion to represent UI wireframes as graphs. Following their solution, we converted the annotation files from our SynZ dataset into graphs.

As we described in Equation 11.1, each UI wireframe can be represented as a list of constituent UI elements represented as a list of vector  $\mathcal{V}$ . This UI wireframe can be represented as a graph ( $\mathcal{G}$ ) where each  $\mathcal{V}$  is modelled as a node, and its relationship, such as position, are represented as edges.

Therefore to convert a UI wireframe to a  $\mathcal{G}$ , we define a rule-base that defines the relationship between a UI element in the UI wireframe with every other element in that UI wireframe. This rule-base considers the position of each UI element with respect to others in four aspects: TLBR (Top-left-bottom-right), TRBL (Top-right-bottom-left), TB (Top-bottom), and LR (Left-Right). Figure 17.1 describes the rule-base equations with sample images. We apply this rule-base for all the UI elements in a UI wireframe to create and fill an adjacency matrix for a given UI wireframe. However, this adjacency matrix captures some unwanted features between UI elements, such as a relationship between two UI elements that are very far apart from each other. Therefore, to improve the features representing the relationship between UI elements in a UI wireframe, we retained only the shortest edges and removed the rest from the adjacency matrix. This final adjacency matrix represents the UI wireframe as the graph  $\mathcal{G}$ . Figure 17.2 shows a sample UI wireframe represented as graph  $\mathcal{G}$ . This

*We transform UI wireframe layout to a graph by modelling UI element category as node and their relative position calculate by a rule-base as edges*

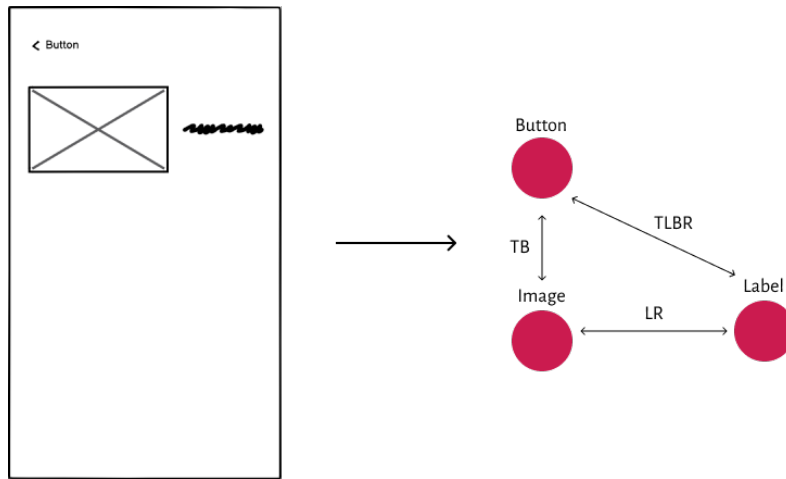


**Figure 17.1:** Rule-based equations for converting different positions of UI elements **A** and **B** from a UI wireframe to an adjacency matrix for UI wireframe graph representation

approach allowed us to capture the UI elements and their relationship in a given UI wireframe layout as a graph representation. We used this graph representation of UI wireframes to train a GCN model.

We loaded the dataset and added a Gaussian noise function to create the misaligned UI layouts as training data. With this training data as input and the aligned UI layout as ground-truth, we created a GCN model using the PyTorch Geometric library developed by Fey et al. (2019). We experimented with two different GCN: three hidden layers with 500 nodes in each layer and ten hidden layers with 50 nodes in each layer. From these experiments, we identified that GCN models did not perform well on the



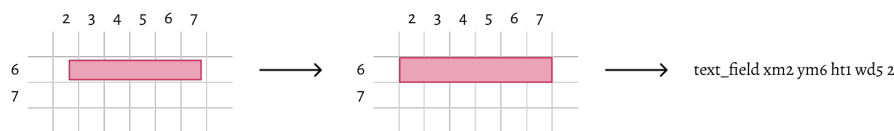


**Figure 17.2:** Sample UI wireframe and its corresponding graph representation

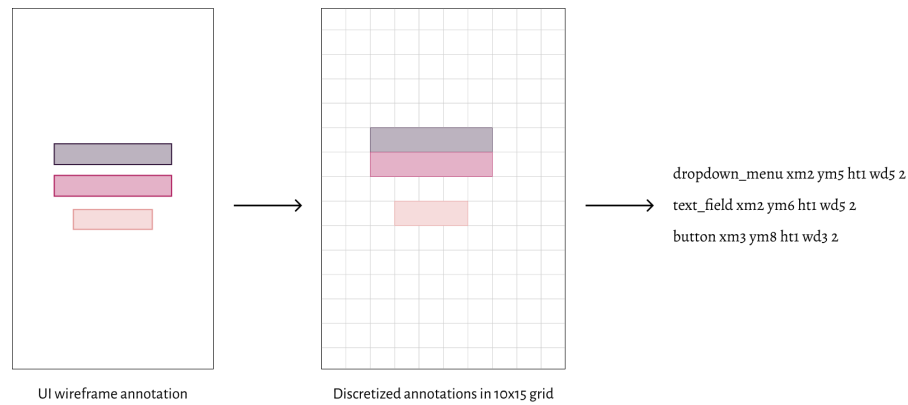
UI layout refinement task. The refined layouts did not correspond well with the ground truth.

### 17.3 DISCRETE SEQUENTIAL REPRESENTATION

Recent research by Gupta et al. (2020) introduced UI layout representation as a sequence of vocabulary to train Transformer architecture. A sequence is defined as an ordered collection of an abstract item. For example, if we consider a paragraph of text in English, we can describe it as a sequence of sentences that can be further split into words. Natural Language Processing DNN tasks like neural machine translation can be achieved by feeding this sequence of English words to a DNN model and training them to output another sequence in the target language like German. Gupta et al. (2020) employed this concept in UI wireframes by considering the constituent UI elements in a given UI wireframe as a sequence of text and used it to train their DNN model to generate and autocomplete UI wireframes. We adopted their approach on representing UI wireframes as a discrete sequence of words to train UI layout refinement models.



**Figure 17.3:** Process of discretizing a UI element bounding box by assigning it to a grid and then converting it to a sequence



**Figure 17.4:** Process of converting a UI wireframe to a sequence

*We transform a UI wireframe layout to a sequence by parsing the UI elements from top-bottom in left-to-right order and concatenating them as sentences.*

As described earlier in [Equation 11.1](#), each UI element can be represented as a vector  $\mathcal{V}$  which describes the category, position, and dimension of the UI element. If we parse the UI elements present in a UI wireframe from top to bottom in left-right order, then we can consider this ordered list as a sequence representing a UI wireframe. This UI element sequence can be further split into sentences that describe each UI element with six words where each word, in order, represents a UI element's category, x, y, height, width, and group ID. Therefore, we can successfully convert a UI wireframe annotation to a sequence without any loss of data.

However, we encountered a significant constraint in using such an approach as the values of x, y, width, and height are in the continuous interval within the range of the UI wireframe dimensions. If the sequence generated using continuous intervals, then the list of all possible sequences becomes infinite. As a solution to this issue, we discretised the UI element position and dimension using a grid, thus reducing the possible outcomes into a finite number.

To discretise the position and dimension of UI elements in a UI wireframe, we split the UI wireframe into a  $10 \times 15$  grid. Then, for a given UI element, we fit its bounding box within this grid: the position is assigned to the top-left corner of the grid in which it overlaps, and the dimension is assigned as all the grids the bounding box overlaps. [Figure 17.3](#) visualises the process of discretising a UI element bounding box values. These discrete values can be used to generate a finite vocabulary of UI element positions and dimensions.

With the discretized UI wireframe annotations, we further convert them to a sequence representation by appending the UI elements present

in a UI wireframe from top to bottom in left-right order into a sequence. Each UI element in this sequence is described as [Equation 17.1](#).

$$c_i \quad x_i \quad y_i \quad h_i \quad w_i \quad (17.1)$$

where for a given UI element  $i$ :

- $c$  = category of the UI element as string (21)
- $x$  = left-most position of the UI element in grid (0-10)
- $y$  = top-most position of the UI element in grid (0-15)
- $w$  = width of the UI element in grid (0-10)
- $h$  = height of the UI element in grid (0-15)

With this process, we can convert a given UI element bounding box into a sequence as shown in [Figure 17.3](#). By applying the discretisation and sequencing process for all constituent UI elements in a UI wireframe and concatenating them, we convert a UI wireframe annotation to a sequence representation. We visualise this process in [Figure 17.4](#)

As the count of UI elements varies per UI wireframe, the sequence length also varies for different UI wireframe. To make this uniform, we limited the maximum number of UI elements per layout to 50; we chose this number based on our exploratory analysis of the SynZ dataset. We padded the shorter sequences with dummy words to make all UI wireframe sequences to a uniform length of 50.

By following these steps, we generate a discrete sequential representation of UI wireframes. Similar to the graph representation, we added Gaussian noise to the UI elements on each UI wireframe to generate the training data. Then, we used the sequence with noise as input and trained a Transformer-Encoder model to output aligned and refined UI wireframe sequences. When fed a UI wireframe sequence, this trained Transformer-Encoder model outputs an aligned and grouped sequence of the given UI wireframe. Unlike GCN, this model provided meaningful alignment and grouping of UI elements in a given UI wireframe.

From the experiments with model architecture and data representation described above, we chose the Transformer-Encoder architecture with discrete sequential data representation for creating RUIITE, a UI layout refiner. In the upcoming chapter, we describe the implementation details, configuration, and training process of the RUIITE.



---

## IMPLEMENTATION

---

In the previous chapter, we described the several experiments that led to the choosing Transformer architecture and discrete sequential representation of UI wireframe for creating RUIITE, a UI layout refiner. In this chapter, we briefly explain the implementation details of RUIITE.

### 18.1 IMPLEMENTATION APPROACH

RUIITE aims to align and group UI elements from a UI wireframe, thus assisting the designer during their prototyping process by creating an aesthetically refined UI wireframe layout.

We applied the traditional supervised deep learning model pipeline outlined by I. Goodfellow et al. (2016) for training RUIITE. In the upcoming sections, we explain the steps in the pipeline in detail.

*From empirical experiments, we chose the Transformer-Encoder architecture for creating RUIITE*

### 18.2 DATASETS

We used the annotations files from the SynZ dataset for training and validating RUIITE. SynZ dataset annotations were extracted and enhanced from the RICO UI screenshots (Section 6.3.1). It contains 58,459 UI annotations with 547,933 UI element bounding boxes for 21 UI element categories. For training and validating RUIITE, we chose a subset of UI wireframes (39,456) from this dataset that does not contain UI elements that collapse to null during discretisation to a grid. We split this dataset into ~90% for training (35,072) and ~10% for validation (4,384).

*We trained and validated RUIITE using 39,456 UI wireframe annotations from the SynZ dataset*

The goal of RUIITE is to refine a misaligned layout. Therefore, to simulate human error during LoFi prototyping and create misaligned layouts, we added noise to the training and validation datasets using three Gaussian, triangular, and uniform distributions. We chose the gaussian distribution noise function with the rationale that UI designers seldom make large er-

*We modelled the misaligned layouts using noise functions sampled from three distributions.*

rors during drag-and-drop prototyping; therefore, to model it, we selected  $\mu = 0$  and some margin of error modelled with  $\sigma$ . Further, to cover other odd scenarios, we used triangular and uniform distributions to generate noise vectors. The three noise function distributions are described with their parameters below.

- **Normal distribution:**  $\mu = 0, \sigma = 1.25 \pm 0.75$
- **Triangular distribution:**  $c = 0, a = b = 1.25 \pm 0.75$  with  $a < b$
- **Uniform distribution:**  $[-x, x]$  where  $x = 1.25 \pm 0.75$

We applied these noise functions to the position and dimension of the UI elements and created the input data. We assigned the data without noise function as the ground-truth predictions. After these steps to create input and ground-truth data, we converted both the training and evaluation dataset into sequence representation as described in the previous section. Further, we removed the sixth word of each UI element sequence to represent grouping information from input sequences with noise; however, the ground-truth sequence contain the expected grouping information. With this approach, we enable the model to learn the grouping along with the alignment of UI wireframe layouts. We used these sequences for training and validating the model

### 18.3 IMPLEMENTATION DETAILS

We used the PyTorch library by Paszke et al. (2019) to create the RUIITE's DNN model. This model only contains the Encoder architecture of a Transformer model, which is then attached to a classification head for predicting the alignment and grouping of the given UI wireframe sequence. Thus, we denote this model further in this dissertation as the Transformer-Encoder model. Figure 18.1 visualizes the architecture of RUIITE's Transformer-Encoder model.

RUIITE's Transformer-Encoder model contains an embedding layer to convert the sequence to a vector representation. It is passed on to a positional encoding layer to store the position of UI element occurrence and ensure the order of UI elements in the sequence. This positional encoded value is further passed through two encoders. Each encoder has a similar architecture with a multi-headed attention model (two heads) and a feedforward network model with 64 dimensions. Finally, the encoded output from the encoders is fed to a classification head with a single linear

layer. A UI wireframe sequence passed through the Transformer-Encoder model as input is processed and returned by the classification head as a sequence of aligned and grouped UI elements.

#### 18.4 CONFIGURATION & TRAINING PROCESS

After several iterations of empirical experiments, we finalised the hyper-parameters of RUIE's Transformer-Encoder model as below.

- **Optimizer:** Adam optimizer with  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$
- **Learning rate:**  $1e-2$
- **Positional-Encoder dropout :** 0.1

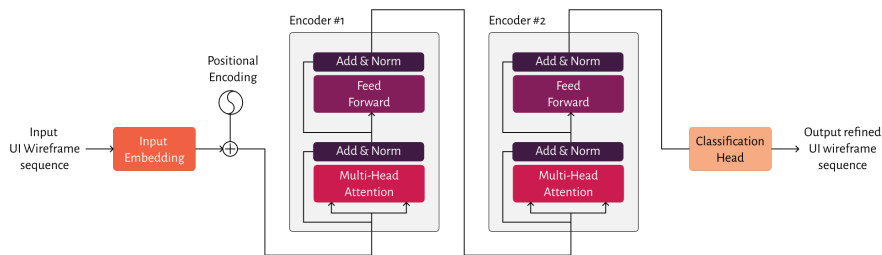
We set the training batch size to 32 to fit the available GPU memory for training the model. We used the cross-entropy loss function to calculate the gradient for backpropagation during training. After every 50 training steps, we calculated the metrics using the validation dataset to monitor the training process. We trained the model for 2,192 steps (2 epochs) and stopped it when the validation loss gradually decreased and plateaued.

For transparency and replicability of this research, we have open-sourced the codebase<sup>1</sup> along with the configuration files.

#### 18.5 POST PROCESSING STEPS

As the Transformer model operates in the sequence domain, the refined UI wireframe output is returned by the model as a sequence. We convert this sequence back to UI wireframe by using the following post-processing steps.

<sup>1</sup> <https://github.com/vinothpandian/RUIE>



**Figure 18.1:** Architecture of RUIE's Transformer-Encoder model

*To convert the  
generated UI layout  
sequence to UI  
wireframe, we created  
a post-processing  
script*

First, we parse the sequence and split it after every six words (category, left-most position, top-most position, height, width, and group). From this sequence, we transform it back to the  $10 \times 15$  discrete grid representation by obtaining the last digits on every word in the sequence. Finally, multiplying the grid representation with its respective grid dimension, we can obtain the post-processed UI wireframe annotation.

With this training and post-processing steps, we obtained the trained model of RUIITE that can align and group constituent UI elements from the given UI wireframe. To understand the quality of UI wireframes refined by RUIITE, we evaluate its Transformer Encoder model. We discuss the evaluation process in the next chapter.



---

## EVALUATION

---

This chapter summarises the evaluation results of the RUI TE UI layout refiner. In the upcoming sections, we briefly describe the evaluation dataset, introduce the metrics used to evaluate RUI TE and report the evaluation results.

### 19.1 DATASET

For evaluating RUI TE, we aimed to use real-life UI layout information. Therefore, we created a script that extracts UI layout annotation from Android smartphone applications<sup>1</sup>. With this script, we captured 500 UI screenshots from an Android smartphone along with their respective layout annotation. Each UI layout annotation contains the alignment and grouping information of all the constituent UI elements in the corresponding UI screenshot. However, as RUI TE is trained with a dataset containing 21 UI element categories, we manually verified the annotations, removed the additional annotations other than the 21 UI elements. Further, for processing with PyTorch, we parsed the annotations CSV format. We used these 500 real-life UI layout annotations as the evaluation dataset.

*We captured 500 UI screenshots from an Android smartphone along with their respective layout annotation for evaluating RUI TE*

### 19.2 METRICS

With this evaluation dataset, we aimed to measure the quality of UI layout refinement achieved by RUI TE. In this section, we introduce the four evaluation metrics we used to measure this information.

**ALIGNMENT SCORE** Gestalt laws are commonly taught and routinely used by designers to create aesthetically pleasing designs. Palmer (1999) describes these laws as a reasonably accurate estimate of how humans

*Alignment score quantitatively measures Gestalt law of alignment*

---

<sup>1</sup> <https://github.com/vinothpandian/capture-android-layout>

perceive structures in a scene or layout. By utilising the gestalt laws, Koch et al. (2016) proposed metrics to compute human perception of interactive layouts quantitatively. As RUI TE aims to align UI wireframe layouts aesthetically, we used the pairwise alignment score metric proposed by Koch et al. (2016) to evaluate the layout quality of RUI TE refined UI wireframes. This score ranges between 0 to 1, and higher scores are considered better.

*Improvement Margin  
measures the margin  
of improvement in  
aligning UI  
wireframes achieved  
by RUI TE*

**IMPROVEMENT MARGIN** The Transformer-Encoder was trained with a synthetically generated noisy input sequence. Therefore, to measure the margin of improvement we attain by aligning this input layout to a refined layout, we introduce the *improvement margin* metric.

To measure the improvement margin, we first calculate the accuracy of the misaligned input sequence with respect to the ground-truth sequence. Similarly, we measure the accuracy of the RUI TE refined layout sequence with the ground truth sequence. By calculating the difference between them (Equation 19.1), we can measure the margin of improvement in aligning UI wireframes achieved by RUI TE.

$$\text{improvement\_margin} = \text{accuracy}_{\text{model\_output}} - \text{accuracy}_{\text{noisy\_input}} \quad (19.1)$$

*Grouping accuracy  
measures the RUI TE's  
accuracy in grouping  
UI elements in a given  
UI wireframe*

**GROUPING ACCURACY** As discussed in the previous section, the input sequence to train the RUI TE model does not contain the sixth word in the UI element sequence representing the grouping information. We removed this information to facilitate the model to learn to group UI elements based on just the UI element category, position and dimensions. In contrast, we keep the grouping information in the ground-truth sequence to compare with the RUI TE's grouping predictions. With this process, we measure the grouping accuracy of RUI TE.

*Mean Average  
Precision (mAP)  
measures the precision  
with which RUI TE  
aligns UI elements in  
a given UI wireframe*

**MEAN AVERAGE PRECISION** COCO detection metrics, proposed by T. Y. Lin et al. (2014), is a commonly used metric to evaluate object detection models. It reports mean Average Precision (mAP) and average recall (AR) of different Intersection over Union (IoU) and area thresholds. IoU measures the overlap between the ground-truth bounding box and DNN model predicted bounding box; it is commonly measured for 50% threshold and in strict mode at 75% threshold. The primary mAP metric is calculated as an average of IoU thresholds ranging from 5% to 95%. Our research concentrates on the alignment and grouping of UI element bounding

boxes; hence, we use the mAP metric over different IoU to evaluate the quality of RUIITE refined UI wireframes. We aim to create the refined UI wireframes to be as close as the ground-truth wireframes; hence, we set the IoU thresholds from 50% to 100% (perfect) overlap.

### 19.3 METHODOLOGY

Similar to the training and evaluation dataset, we transformed the evaluation dataset into a sequence using the same vocabulary as the training and evaluation dataset. These sequences act as the ground truth for evaluating RUIITE. Then, we added random noise to these sequences, similar to the training dataset, to create the evaluation input to RUIITE. Finally, we loaded the final trained Transformer-Encoder model, sent the evaluation input data, and gathered the output sequence for calculating the evaluation metrics. We used Python scripts to measure the alignment score, improvement margin, grouping accuracy and mAP of RUIITE. We report the results and discuss them in the upcoming section.

### 19.4 RESULTS

RUIITE provided satisfactory results on almost all evaluation metrics. In this section, we discuss the results of all the metrics successively.

**ALIGNMENT SCORE** On the gestalt law based alignment score metric proposed by Koch et al. (2016), RUIITE provided a score of **0.87**. This result shows that RUIITE performs adequately in aligning the misaligned UI wireframes.

**IMPROVEMENT MARGIN** Further, to understand the improvement in alignment accuracy, we calculated the improvement margin metric. From this analysis, we found that the accuracy of misaligned inputs compared to ground-truth was 0.37 and the accuracy of RUIITE refined layouts with ground-truth was 0.75. By applying the formula, we measure the improvement margin score as 0.38. This score shows that RUIITE improves the visual alignment of a UI wireframe from a misaligned layout by **~38%**.

**GROUPING ACCURACY** In addition to alignment, RUIITE also performs the grouping of UI elements. To measure the grouping quality, we calculated the grouping accuracy. Here RUIITE provides a grouping accu-

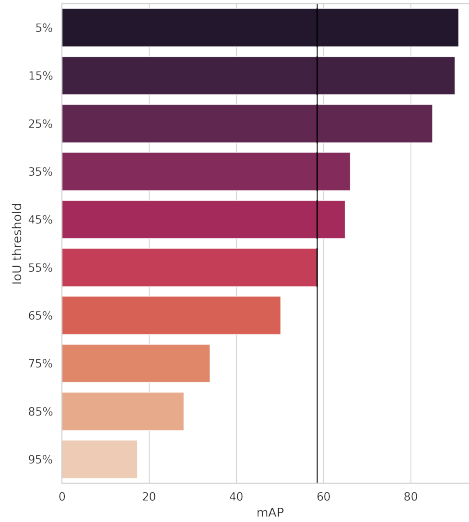
racy of **0.2**. From this result, we understand that the RUIE model does not adequately group UI elements as anticipated. We aim to investigate further and improve this feature of RUIE in our future work.

**MEAN AVERAGE PRECISION** Lastly, RUIE provides 58.53% mAP for refining misaligned UI wireframes. In strict mode, 75% IoU threshold as per COCO detection metrics by T. Y. Lin et al. (2014), RUIE provides 33.99% mAP. Further, Table 19.1 and Figure 19.1 shows that the mAP metrics are adequate for IoU thresholds 35% to 65%.

Surprisingly, for higher thresholds, RUIE only provides ~20% mAP. This low accuracy for higher thresholds shows that the model does not overfit the data and instead learns to align through the training process.

IoU threshold	mAP
5%	90.96
15%	90.13
25%	84.99
35%	66.11
45%	64.94
55%	58.67
65%	50.20
75%	33.99
85%	27.99
95%	17.31
Mean	58.53

**Table 19.1:** mAP scores of RUIE for different IoU threshold values



**Figure 19.1:** mAP score of RUIE model during evaluation for different IoU thresholds ranging from 5% to 95%

Overall, RUIE provides satisfactory results for almost all metrics except for grouping. These results show that RUIE refines misaligned UI wireframes with moderate quality. To further understand the designer satisfaction in using the AI assistance provided by RUIE during LoFi prototyping, we conducted a quantitative user evaluation along with a qualitative semi-structured interview. We discuss these user evaluations in the upcoming chapter.

# 20

## USER EVALUATION

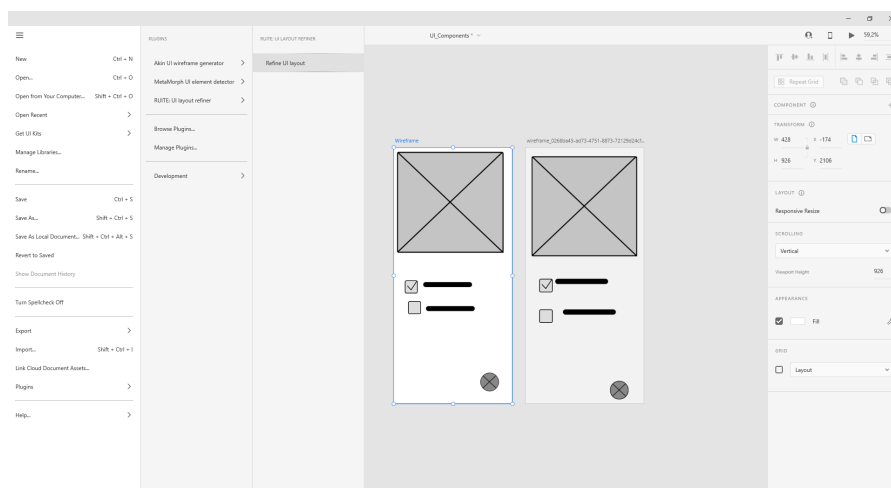
The evaluation metrics described in the previous chapter shows that RUI TE refines UI elements in a given UI wireframe with adequate results. Further, to understand the designer satisfaction in using AI assistance from RUI TE during LoFi prototyping, we conducted a quantitative and qualitative study. We describe these study setup, methodology and results in this chapter.

### 20.1 WEB API & ADOBE XD PLUGIN

Similar to other tools in the BlackBox toolkit, using RUI TE, we intend to provide a modular and reusable interface for refining UI wireframes by creating a Web API. We developed this API in Python using PyTorch and FastAPI<sup>1</sup> frameworks.

*We created a web API for RUI TE to create a modular and reusable interface*

<sup>1</sup> <https://fastapi.tiangolo.com/>



**Figure 20.1:** Screenshot of RUI TE plugin and the refined wireframe in Adobe XD prototyping tool

This Web API<sup>2</sup> accepts a UI wireframe layout annotation as JSON in the HTTP request, refines the layout using RUIITE and responds with the refined UI wireframe layout annotation in JSON format.

To use this Web API, we developed a plugin for Adobe XD<sup>3</sup> prototyping tool. This plugin enables users to access RUIITE directly from the Adobe XD prototyping application. Therefore, a designer can drag-and-drop to create UI wireframes in Adobe XD and to align the wireframe by calling RUIITE directly from the plugin menu. Figure 20.1 shows a screenshot of the RUIITE plugin in Adobe XD. We used this setup for conducting our user study.

## 20.2 METHODOLOGY

*We used the standard  
After-Scenario  
Questionnaire (ASQ)  
for measuring user  
satisfaction in using  
RUIITE*

To assess the designers' satisfaction in utilizing AI during their traditional prototyping process by refining UI wireframes while prototyping, we conducted a quantitative study using After-Scenario Questionnaire (ASQ), created by Lewis (1991b), followed by semi-structured interviews (15 min) to obtain qualitative feedback. The ASQ measures user satisfaction on the following three scales.

Q1 Ease of completing the tasks

Q2 The amount of time it took to complete the tasks

Q3 The supporting information provided while completing the tasks

## 20.3 PARTICIPANTS

We used purposive and snowball sampling to recruit 10 participants: 5 UI/UX designers and 5 HCI grad students (8 male and 2 female). The participants were between 24 to 32 years of age and had 1 - 3 years (Mean = 1.6, SD = 0.84) of prior prototyping experience.

### 20.3.1 Scenario & tasks

The participants were given a scenario to interact with the RUIITE using the Adobe XD plugin to refine LoFi prototypes. They were instructed to complete the following three tasks.

<sup>2</sup> <https://ruiite.blackbox-toolkit.com/>

<sup>3</sup> <https://www.adobe.com/products/xd.html>

1. Create a LoFi prototype by drag-and-dropping UI wireframe elements from AdobeXD components
2. Instead of aligning manually, use the RUIITE plugin to align the wireframes
3. Connect the wireframes into a UI prototype

We asked participants to create and refine multiple UI wireframes as preferred until they are satisfied during this study. Task 1 was randomized with a different set of screens for different participants.

### 20.3.2 *Material & Apparatus*

Participants performed the aforementioned tasks remotely. We connected the participants via video conferencing and provided them access to our computer, which already contains Adobe XD with RUIITE plugin installed.

### 20.3.3 *Procedure*

Due to the ongoing COVID-19 pandemic, we conducted this study remotely via video conferencing. Once the participant agreed to participate in the study, we acquired their informed consent and then invited them to a virtual meeting. Before the study, we briefed the participants about the study's purpose and the assigned task. Then, participants were given time to explore the RUIITE plugin and Adobe XD features to familiarize themselves with the environment and test the remote control setup.

After this brief exploration period, participants performed the assigned tasks and provided feedback using a think-aloud protocol. Once the participant completed the tasks, they were asked to fill the ASQ. On average, participants took around 15 minutes to complete the tasks and fill the questionnaire. After the study, we conducted semi-structured interviews (~15 mins) to understand the participant's view on the AI assistance during the LoFi prototyping approach and better understand and interpret the ASQ results.

*Study was conducted remotely with screen sharing*

### 20.3.4 *Results & Discussion*

RUIITE received an above-average satisfaction level for all three questions in the ASQ: ease of completing the tasks (5.3), the amount of time it took to

**Table 20.1:** Results of the ASQ study with participant's prior experience in UI prototyping

ID	Prototyping experience (in years)	Q1	Q2	Q3
P1	3	6	7	6
P2	2	6	6	7
P3	3	5	7	7
P4	2	6	7	6
P5	1	4	6	6
P6	1	3	7	6
P7	1	5	7	5
P8	1	6	6	6
P9	1	7	7	7
P10	1	5	6	6
<b>Mean</b>		5.3	6.6	6.2
<b>SD</b>		1.16	0.52	0.63

complete the tasks (6.6), and the supporting information provided when completing the tasks (6.2). [Table 20.1](#) summarizes the ASQ score from each participant along with their UI prototyping experience.

We further analyzed the correlation between the experience of the participant and their ASQ satisfaction score. Unlike the ordinal values of the 7-point Likert scale in ASQ, the prototyping experience of a UI designer is on the ratio scale. Therefore, we converted the experience to an ordinal scale using average ranking and used rank correlation coefficient by Spearman (1904) to measure the correlation between them.

Surprisingly, we found a low positive correlation for all three ASQ question score and years of experience:  $r_{\text{experience} \times Q1} = -0.273$ ,  $r_{\text{experience} \times Q2} = 0.323$ , and  $r_{\text{experience} \times Q3} = 0.394$ . This result shows that the experience of participants does not significantly impact their satisfaction in using AI assistance during LoFi prototyping. [Figure 20.2](#) visualizes the correlation trend between precision and the three questions from ASQ. To understand this trend, we further analyzed our semi-structured interview results.

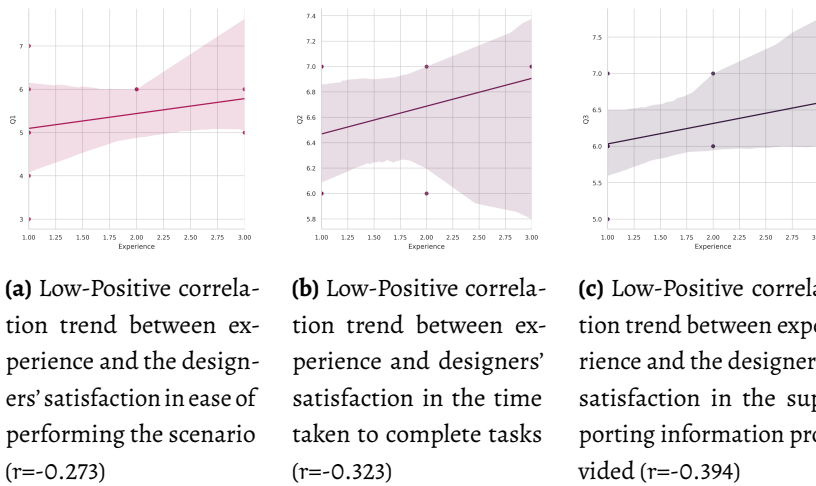
The qualitative feedback showed that both novice and experienced participants preferred refinement of UI wireframe using AI in a button-click. Further, the participants found the high automation without compromising their autonomy to be helpful. They preferred to keep grid lines for

*RUIITE received an above-average satisfaction level for all three questions in the ASQ*

*We found the user satisfaction of both novices and experienced designers are roughly similar while using RUIITE*

*Both novice and experienced participants preferred refinement of UI wireframe using AI in a button-click.*





**Figure 20.2:** Spearman rank correlation between UI element detection precision and the three questions from ASQ

manual alignment and sporadically using one-click UI refinement using RUIITE.

“ I would definitely prefer this technique if the system somehow knows exactly what I want to do with the layout. If the system works perfectly, I would use it everyday for prototyping.

-P1

“ The system should take some input constraints from the user. I would prefer a hybrid of both user constrained and automated alignment.

-P6

“ Having an extra button wouldn't complicate my workflow. Whether I use it or not should be my choice, depending on what layout I create.

-P9

Overall, all participants generally perceived using AI as an assistant during LoFi prototyping as a helpful approach that simplifies the traditional prototyping process. From the qualitative feedback, we understand that most participants were willing to incorporate RUIITE in their workflow and suggested few exciting features for improving RUIITE. The upcoming chapter summarises our findings while creating and evaluating RUIITE and describes the identified limitations and future work.



---

## SUMMARY & FUTURE WORK

---

This part of the thesis introduced RUI TE, a UI wireframe refiner. RUI TE assists UI designers during the LoFi prototyping process by aligning and grouping the UI wireframe using a Transformer-Encoder model. We trained this model with 35,072 UI annotations from the SynZ dataset. Upon evaluation with 500 real-life UI layouts, RUI TE's model provides adequate results with almost all metrics: alignment score (0.87), improvement margin (~38%), grouping accuracy (20%), and mean Average Precision (58.53%). Further, we conducted a quantitative study using After-Scenario Questionnaire (ASQ) followed by semi-structured interviews to understand the designer satisfaction using RUI TE. The results indicated that designers experience an above-average satisfaction level towards ease of task completion (5.3), time taken (6.6), and supporting information (6.2) upon utilising AI assistance during LoFi prototyping to align and group the UI elements in their UI wireframe. The qualitative feedback indicated that both novice and experienced participants preferred refinement of UI wireframe using AI in a button-click.

RUI TE tool addresses a part of RQ 2, 3 and 4 of this thesis. In addition, through RUI TE, we endeavoured to fulfil the identified research gaps. Moving from the prior research on deterministic models, we utilised the self-attention mechanism of Transformer-Encoder architecture to refine a given UI layout. Further, we evaluated our system with quantitative and qualitative studies to uncover the satisfaction of designers in using AI as assistance during the LoFi prototyping process.

However, by analysing the feedback we received during user evaluation, we identified the following limitations with our current system, which we would improve in our future work.

From the qualitative feedback with users, we understood the need for further fine-grained alignment and grouping control during UI prototyping. We plan to facilitate users to specify constraints on UI elements that

RUIE can alter. We would also like to enable RUIE to understand the context of text within the UI elements for alignment and grouping.

A significant limitation of our current implementation is that RUIE does not group UI elements with adequate accuracy (~20%). We would like to improve this metric by experimenting with alternative DNN architectures and retraining RUIE with a larger dataset.

## PART V

# METAMORPH

In this final part, we describe MetaMorph, a UI element detector that assists UI designers *after LoFi prototyping*. MetaMorph assists UI designers by detecting the constituent UI elements of LoFi sketches and their location and dimension; thus, enabling the transformation of LoFi sketches to higher-fidelities. To train MetaMorph, we used synthetic LoFi sketches Syn, SynZ and further fine-tuned the model with the real-life LoFi sketches from the LoFi sketch dataset. MetaMorph provides 47.8% mAP for hand-drawn LoFi sketches. Further, the results from the After Scenario Questionnaire to assess user satisfaction indicates that designers experience an above-average satisfaction level towards ease of task completion (4.9), time taken (5.3), and supporting information (5.3) upon utilizing AI assistance for transforming lo-fi sketches. Their qualitative feedback indicates that they perceive utilizing AI as a novel and useful approach to transform LoFi sketches into higher fidelities.



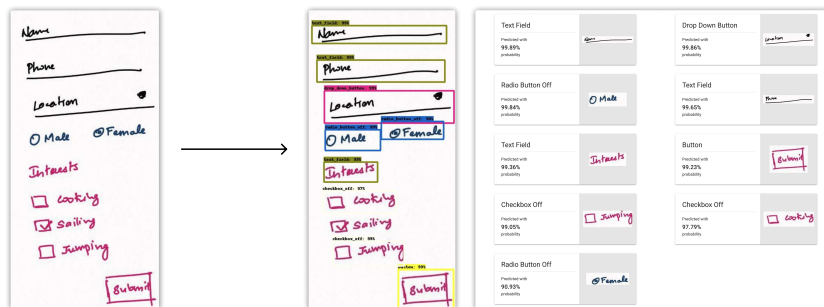
<https://metamorph.blackbox-toolkit.com>



## BACKGROUND & PROPOSED SOLUTION

MetaMorph is a UI element detector that detects the constituent UI elements of LoFi sketches, thereby supporting designers at the end of the LoFi design process by enabling them to transform LoFi prototypes to higher fidelities. This part of the dissertation was published at CHI 2019 (Suleri, Pandian, et al., 2019), OzCHI 2020 (Pandian, Suleri, Beecks, et al., 2020) and IUI 2021 (Pandian, Suleri, and Jarke, 2021a). This chapter briefly summarises the existing literature in automating LoFi transformation, reemphasises the research gaps in the existing literature, and introduces our proposed solution to tackle this problem.

*MetaMorph assists designers **after** their LoFi prototyping task by enabling them to transform LoFi prototypes to higher fidelities.*



**Figure 22.1:** MetaMorph takes a LoFi sketch (left) as input and detects UI elements (right) using DNNs. For each detected UI element, it provides the object's category, prediction probability, and bounding box details (x,y coordinates of the top left corner, width, and height).

### 22.1 BACKGROUND

MetaMorph assists UI designers at the end of the UI design process by automating the prototyping fidelity transformation. As discussed earlier, UI designers follow the traditional prototyping process of ideating with LoFi prototypes, manually creating and reiterating MeFi and HiFi prototypes

to evaluate their system. To improve this process and support designers to quickly and efficiently create realistic higher fidelity prototypes, many researchers attempt to automate the prototyping process using machine intelligence.

*Most of the prior research on automating LoFi transformation was conducted with classical pattern recognition, and recent research explored object detection DNN models.*

To briefly summarize the related work from our literature review ([Section 2.3](#)), most of the early works in automating LoFi transformation to higher fidelities, conducted by Coyette et al. (2004), Landay (1996), Perez et al. (2016), and Plimmer et al. (2003), use classical pattern-recognition algorithms to transform LoFi sketches to code. These solutions are engineered to classify a limited number of UI elements. As LeCun et al. (2015) explains, adapting such pattern-recognition algorithms is demanding as there is no general method for extracting a good set of features for a specific problem. Therefore, recent proof-of-concept projects, created by Ashwin (2018), Benjamin (2017), Microsoft AI Labs (2018), and Tony (2017), use DNNs to tackle similar object detection tasks.

## 22.2 IDENTIFIED GAPS

*Prior research and projects do not measure the impact nor the utility of utilising AI to automate the transformation of LoFi to higher fidelities*

However, these solutions, either pattern recognition based or deep learning based, only concentrate on converting LoFi sketches to code; they explore neither the impact nor the utility of utilising AI to automate the transformation of LoFi to higher fidelities through systematic research. They also do not provide a generalizable UI element detector as they use either machine-generated sketches (Ashwin, 2018) or a tiny dataset for training (149 sketches) (Microsoft AI Labs, 2018). Therefore, there is a clear need for systematic research on utilising AI assistance to transform LoFi to higher fidelities and understanding designers' satisfaction upon using AI assistance during the UI prototyping process.

## 22.3 PROPOSED SOLUTION

This part of the dissertation aims to bridge the identified research gap by conducting systematic research to understand the designers' need for automatic prototyping fidelity transformation. We designed MetaMorph following the principles of a traditional deep learning pipeline. We started the development process by collecting and refining datasets ([Chapter 4](#)), then using it to train and evaluate a Deep Neural Network (DNN). Finally, We conducted user satisfaction study and semi-structured interviews with UI/UX designers to understand their satisfaction level using such



a tool. Further, we made the system modular and pluggable with any future research project or product, thus allowing designers to take complete control of the transformation process of the LoFi prototype; thereby, enabling UI designer to convert LoFi to either MeFi or HiFi based on their preference.

Before implementing MetaMorph, we experimented with various data representation and DNN model architectures. Recent advancements in object detection provided us with varied alternatives and approaches in creating a UI element detector. A LoFi sketch can be represented as a sketch and an annotation file with a list of constituent UI elements in that sketch along with their location and dimension in that image or as a list of UI element sketch images. As there is no prior study that experimented with the different alternatives of data representation and model architectures, we conducted empirical experiments by training different object detection model architectures with varied hyperparameters and loss functions. We utilised UISketch, Syn, and SynZ datasets (Chapter 4, Chapter 6) to train the MetaMorph UI element detector and the LoFi sketch dataset to evaluate the UI element detection model.

In the upcoming chapter, we discuss the data representation and model architecture experiments in brief to justify the final model architecture of MetaMorph.

**Model:** SSD Resnet50  
(RetinaNet)

**Datasets:** Syn, SynZ  
and LoFi sketch

**Evaluation:** Two AI  
metrics, a quantitative  
and a qualitative user  
evaluation with  
UI/UX designers



Szegedy, Toshev, et al. (2013) defines *object detection* as locating and classifying an object from a variable set of objects. It involves two different tasks: *localization*—determining the location and dimension (bounding-box) of the object; and *classification*—classifying the object into one of many different predefined categories. Recent advancements in object detection provide us with varied alternatives and approaches in creating a UI element detection model. The goal of MetaMorph, our UI element detection model, is to identify the UI element category and its corresponding location and dimension. We conducted several experiments with various data representations that fit various DNN model architectures before finalizing MetaMorph. This chapter briefly discusses the experiments and the design rationale in the model architecture and parameter choices in the MetaMorph model.

### 23.1 MODEL ARCHITECTURES

From literature review, we identified two leading architectures for objection detection: Region-Proposal Networks (RPN) and Single-Shot Detection (SSD) networks (J. Huang et al., 2016).

*We experimented with four model architectures to create MetaMorph*

#### 23.1.1 Region Proposal

The RPN architecture consists of either a pattern-recognizer region-proposal algorithm coupled with a classification DNN or an RPN DNN and classification DNN in sequence. The pattern recognizer region-proposal algorithm or RPN DNN is solely used to detect Regions of Interest (RoI) from a given image. These proposed RoIs are then passed to the second classification DNN, which categorizes the object into a set of predefined object categories; thus, identifying the location, dimension, and category

*Region Proposal architecture contains two models: a region proposal algorithm or DNN and a classification DNN*

of all the objects in a given image. The state-of-the-art DNNs with RPN architecture are RCNN by R. Girshick et al. (2014), Fast RCNN by R. B. Girshick (2015) and Faster RCNN by Ren et al. (2015). RCNN and Fast RCNN uses the selective search pattern recognition algorithm by Uijlings et al. (2013) for RoI proposal, whereas Faster RCNN uses an RPN DNN for RoI proposal and classification.

### 23.1.2 Single-Shot Detection

*Single-Shot Detection models contain only one DNN, which is end-to-end trainable for object detection*

The Single-Shot Detection (SSD) architecture consists of a single DNN model trained end-to-end to locate and categorize objects in an image. This model is trained to split the image into RoIs and classify the objects from the RoI in a single training epoch. The state-of-the-art models which use this approach are YOLO by Redmon et al. (2016) and SSD by W. Liu et al. (2016).

The survey by J. Huang et al. (2016) on object detection models compared the speed and accuracy of both these architectures. This comparison claims that Faster RCNN, an RPN architecture, provides better accuracy than SSD architecture; while, SSD provides faster prediction results than others.

Apart from these overarching differences in model architectures, the object detection models also differ on the neural network layers and the layer block architectures such as Fully Convolution Network (FCN), U-Net, and Feature Pyramid Networks (FPN). Also, the backbone classification networks might change in the RPNs. The most popular backbones are ResNet, MobileNet and Inception ResNet. Similarly, as an improvement in SSD models T.-Y. Lin et al. (2017) proposed RetinaNet using a new loss function, focal loss. T.-Y. Lin et al. (2017) claims that RetinaNet provides better accuracy than other models while providing quicker results.

## 23.2 DATA REPRESENTATIONS

*We experimented with two data representation of LoFi sketches to train and evaluate MetaMorph*

Apart from the differences in model architectures, the data representation also differs for different model architectures. For the earlier RPN models, with pattern recognizer based region proposal algorithms, the backbone classifier can be trained with individual UI element sketch representation such as the UISketch dataset and coupled with an external RoI proposal system.

In contrast, in the recent RPN and SSD architectures, a LoFi sketch can be represented as an image and its respective annotation file. This

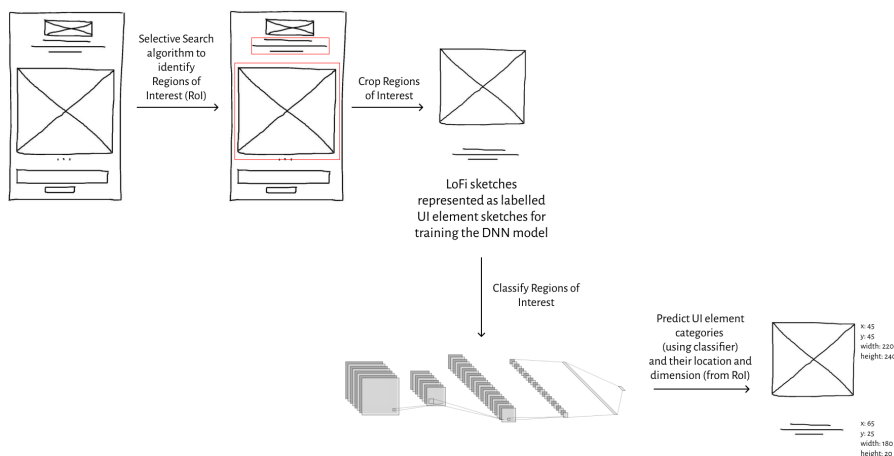
annotation file contains the list of UI element categories present in the LoFi sketch and their bounding box position in the 2D axis (x and y) and dimension (width and height).

To answer RQ2, using these ideas, we represented LoFi sketches in two different ways to train different model architectures before finalizing an apt representation and architecture for creating MetaMorph. We discuss these representations in the upcoming subsections.

### 23.2.1 Labelled UI element sketches

To train the pattern recognizer based region proposal architecture models, we represent LoFi sketch data as a collection of labelled UI element sketches from the UISketch dataset. These UI element sketches were used to train a DNN model using the supervised learning technique. We then use this trained DNN model as a backbone classifier and couple it with a region proposal algorithm to create a UI element detection model. We reused the ResNet-152 model that performs best in machine recognition study (Chapter 9) as the backbone classifier. For region proposal, we used the *selective search* algorithm proposed by Uijlings et al. (2013) in fast mode for region proposal. Similar to R. Girshick et al. (2014), we limited the region proposals to 2000 regions per LoFi sketch. Using this architecture and data representation, we trained both RCNN and Fast RCNN models. From our experiments, we observed that these models performed poorly in the UI element detection task. Although the classifier performs with 91.7% accuracy, the region proposal algorithm using a selective search

*We represent a LoFi sketch as a collection of labelled UI element sketches for training an algorithmic region proposal model with a UI element classifier DNN.*



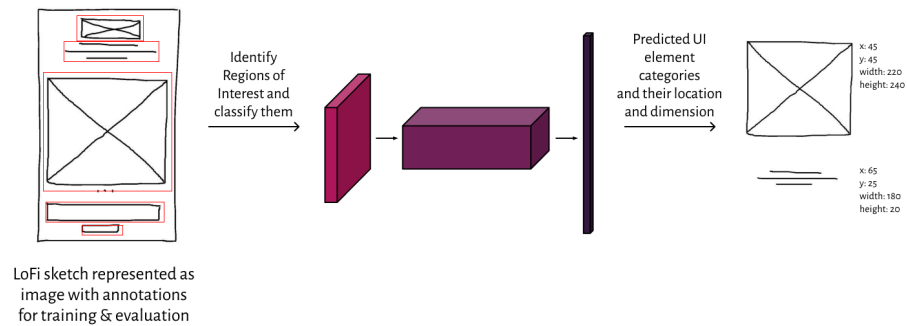
**Figure 23.1:** LoFi sketches represented as labelled UI element sketches for training the classification DNN

algorithm did not adequately identify the possible regions. Hence, the overall detection was inadequate for the UI element detection task.

### 23.2.2 LoFi sketch with annotations

*We represent LoFi sketches as a LoFi sketch image with its corresponding annotation file*

Widely used datasets such as Pascal VOC by T. Y. Lin et al. (2014) and COCO dataset by Everingham et al. (2010) for object detection represents an image as a series of objects and their annotations as bounding boxes. Similarly, we represent LoFi sketches as a LoFi sketch image with its corresponding annotation file. This annotation file, either in CSV or JSON format, contains a list of UI element categories present in this image, along with its location and dimension.



**Figure 23.2:** LoFi sketches represented as an image and annotation file with the list of constituent UI element categories, their location and dimension

To generate such a dataset of LoFi sketches with annotation files, we generated synthetic datasets, Syn and SynZ, using the UISketch dataset (Chapter 6). Using these datasets, we trained Faster R-CNN Resnet by Ren et al. (2015), SSD MobileNet by W. Liu et al., 2016, and SSD Resnet (RetinaNet) by T.-Y. Lin et al. (2017) detection models with various hyperparameters. We observed that these models performed better in UI element detection than the region proposal algorithm and classifier models; however, training the system was slower due to high memory and GPU consumption.

Based on these empirical trials, we chose SSD Resnet 50 (RetinaNet) architecture for UI element detection. RetinaNet is an improvement on single-shot detection models using a new loss function, focal loss. It provides better accuracy than other models while providing quicker results.

In the upcoming chapter, we describe the implementation details, configuration, and training process of the MetaMorph UI element detector.

# 24

---

## IMPLEMENTATION

---

As discussed in the previous chapter, after several experiments with model architectures and data representations, we used SSD Resnet (RetinaNet) for training the MetaMorph UI element detector. In this chapter, we discuss the training datasets, configuration, and training process.

### 24.1 IMPLEMENTATION APPROACH

The goal of MetaMorph is to assist in the transformation of lo-fi to higher fidelities by providing an initial step. MetaMorph is an object detector fine-tuned to locate and classify UI elements from lo-fi sketches. It performs two functions: localization and classification. As Szegedy, Toshev, et al. (2013) describes, *Localization* is identifying the object's location in an image. It returns bounding box details (x,y coordinates of the top left corner, width, and height) of an object's position in the image. *Classification* is identifying the object's category from a set of predefined categories.

To create MetaMorph, we followed the supervised deep learning model pipeline outlined by I. Goodfellow et al. (2016). As a supervised deep learning model learns to recognize and classify based on the underlying ground truth data, we used our synthetically generated LoFi datasets Syn and SynZ for pretraining the UI element detection model. We further fine-tuned the pretrained model with real-life LoFi sketches from our LoFi sketch dataset to create MetaMorph.

*From empirical experiments, we chose the SSD Resnet (RetinaNet) model for creating MetaMorph*

### 24.2 DATASETS

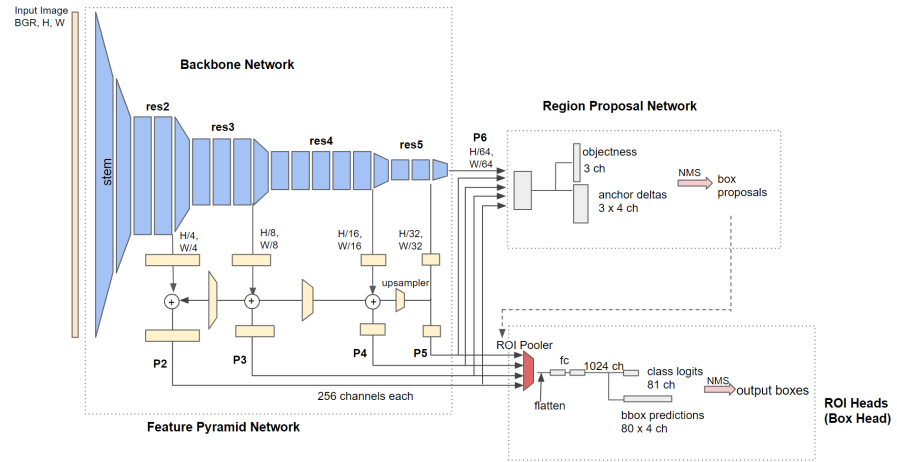
As discussed in [Chapter 6](#), we synthetically generated 300,377 LoFi sketches and their annotation files (combined both Syn and SynZ datasets) to train the object detection model of our UI element detector. In addition, we collected 4,527 hand-drawn LoFi sketches (LoFi Sketch dataset) for training

*We trained MetaMorph using LoFi sketches from Syn, SynZ, and LoFi sketch dataset*

and evaluating the UI element detector. We used all three datasets to train MetaMorph and 20% of the LoFi sketch dataset to evaluate the final trained model.

### 24.3 IMPLEMENTATION DETAILS

We implemented MetaMorph in the python programming language using PyTorch package by Paszke et al. (2019) and Detectron2 library by Wu et al. (2019). We reconfigured the Retinanet model from the Detectron2 library and used it for creating the UI element detector. Figure 24.1 created by Honda (2020) shows the detailed architecture of RetinaNet model from the Detectron2 library. After pretraining, we used the LoFi sketch dataset to fine-tune further and create the MetaMorph.



**Figure 24.1:** Architecture of the SSD Resnet 50 (Retinanet) model from the Detectron2 library as depicted by Honda (2020)

### 24.4 CONFIGURATION

The Retinanet model from Detectron2 library was pre-trained with COCO dataset collected by T. Y. Lin et al. (2014). Therefore, we loaded the pre-trained model checkpoints and fine-tuned the object detection model with the Syn and SynZ datasets to detect UI element sketches.

We set the training batch size to 64 to fit the available GPU memory. We removed all the preprocessing steps provided by the model, as the images in the synthetic dataset were preprocessed and augmented during



generation. We assigned the number of categories as 21, which denotes the number of UI element categories in our dataset.

We determined the following hyperparameters after multiple iterations of empirical experiments on the training data for training the RetinaNet object detection model:

- **Feature extractor:** Resnet model with 50 layers
- **Image rescaling:** Rescaled to a square with 640 pixels on each side
- **Optimizer:** Stochastic Gradient Descent optimizer with cosine learning rate decay
- **Learning rate:** Starts at 0.0399 and decays with cosine learning rate decay strategy
- **Anchor boxes aspect ratios:** 0.5, 1, 2
- **Anchor box sizes:** 32, 64, 128, 256, and 512 (interpolated with 0.25 and 0.5 anchor box sizes)

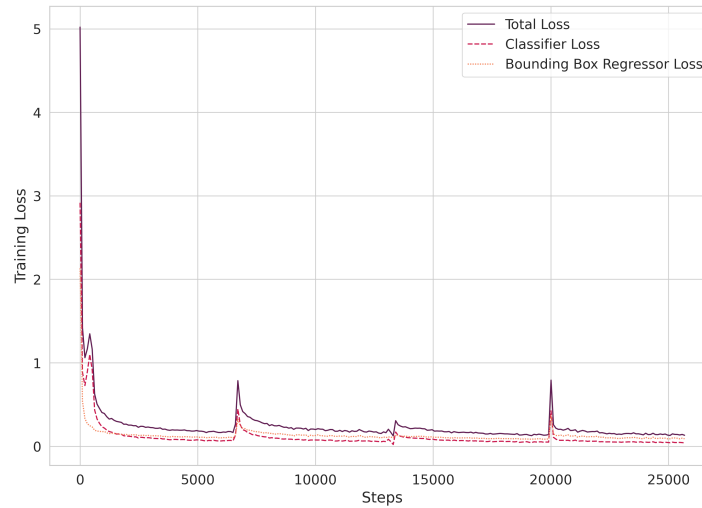
For transparency and replicability of this research, we have provided the codebase along with the configuration file <sup>1</sup>.

## 24.5 TRAINING PROCESS

We trained the Retinanet model in two steps: pre-train the model with Syn and SynZ datasets, then fine-tune with the LoFi sketch dataset. We followed this approach as the synthetic dataset approximates the real LoFi sketches but does not accurately depict the real-life hand-drawn LoFi sketches. Therefore, pretraining with the synthetic dataset and further fine-tuning with the real-life dataset provides the most stable results.

We set up the training pipeline for the pretraining task with the Syn and SynZ datasets and created a GPU cluster batch job to train the object detection model. We used two NVIDIA Tesla P100 16GB GPUs for training these models. We trained the model for 25,679 steps (~6 epochs). We stored the model checkpoint after every 1,000 steps and restored training if the model fails due to GPU cluster timeout. Figure 24.2 shows the bounding box regressor loss, classifier loss and total loss values of the Retinanet model during training with Syn and SynZ dataset. The intermediate peaks are the steps where we resumed training after GPU cluster timeout failure.

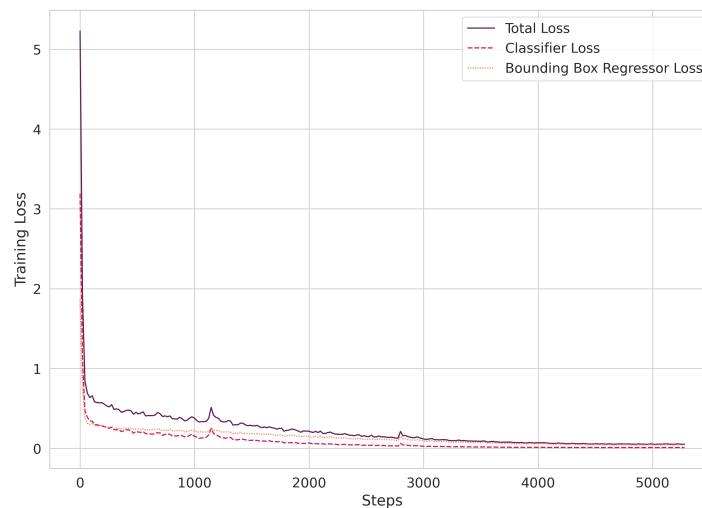
<sup>1</sup> <https://github.com/vinothpandian/SynZ>



**Figure 24.2:** Training loss per steps of the RetinaNet model during training with the LoFi sketches from the Syn and SynZ training datasets

As visualized in [Figure 24.2](#), the general trend of the training loss of the RetinaNet model is in a downward curve and reaches a point of stability.

We loaded the pre-trained weights obtained after training the model with the Syn and SynZ datasets for further fine-tuning with the LoFi sketch dataset. We then continued training for 5,279 steps (~46 epochs) until the loss curve plateaued. [Figure 24.3](#) shows all the loss values of the final fine-tuning of the Retinanet model during training with the LoFi sketch dataset. Similar to the trend of training loss with the Syn and SynZ dataset, the [Figure 24.3](#) shows training loss decreases in a downward curve towards a plateau of stability.



**Figure 24.3:** Training loss per steps of the RetinaNet model during training with the 3,632 LoFi sketches from the LoFi sketch training dataset

With this training process, we obtained the trained weights of the UI element detection model: MetaMorph. Using the 20% of real-life hand-drawn LoFi sketch evaluation dataset that we set aside, we further evaluated the MetaMorph UI element detection model to measure its precision and recall. We discuss the evaluation of our UI element detector in the next chapter.



# 25

---

## EVALUATION

---

In this chapter, we discuss the evaluation results of the MetaMorph UI element detector. In the upcoming sections, we describe the evaluation dataset and introduce the evaluation metric used to measure the precision and recall of MetaMorph and briefly discuss the evaluation results.

### 25.1 DATASET

To evaluate the ecological validity of MetaMorph's object detection model, we aimed to evaluate it with hand-drawn lo-fi sketches. We evaluated the MetaMorph UI element detection model with 895 real-life hand-drawn LoFi sketches (20%) from the LoFi sketch dataset. These sketches were collected from UI/UX designers, front-end developers, HCI and CS grad students; thus, they are representative of LoFi sketches encountered in the real world. [Table 25.1](#) shows the distribution of UI element categories in the evaluation dataset.

*We evaluated MetaMorph with 895 real-life hand-drawn LoFi sketches from the LoFi sketch dataset*

### 25.2 METRICS

We used this evaluation dataset to measure the average precision and recall of the trained UI element detection model. According to the survey of Padilla et al. ([2021](#)), the measures in COCO detection metrics proposed by T. Y. Lin et al. ([2014](#)) is one of the commonly used evaluation measures for object detection models. Therefore, we used COCO detection metrics to calculate the evaluation results.

*Mean Average Precision (mAP) measures how accurate MetaMorph detects UI elements in a LoFi sketch*

The central quantitative measures used in COCO detection metrics are the Mean Average Precision (mAP) and Average Recall (AR). These metrics are then further separated into different Intersection over Unions (IoUs) and the bounding box area. The IoU measures the amount of overlap

*Average Recall (AR) measures how good MetaMorph is in finding all the correct UI elements.*

**Table 25.1:** Distribution of UI element category in the 895 LoFi sketches in the LoFi sketch evaluation dataset

Category	UI element category	Count
Composite	Card	551
	Alert	57
	Menu	48
	Chip	19
	Grid list	12
	Data table	3
	Tooltip	1
Standalone	Label	3280
	Image	2457
	Button	809
	Text field	430
	Checkbox checked	103
	Switch enabled	100
	Floating action button	97
	Checkbox unchecked	50
	Switch disabled	48
	Text area	14
	Radio button unchecked	10
	Dropdown menu	6
	Slider	3
	Radio button checked	2

between the ground truth bounding box and the predicted bounding box. Overall, the COCO detection metrics reports 12 values.

### 25.3 METHODOLOGY

We preprocessed the 895 LoFi sketches from the evaluation dataset using the OTSU thresholding and binary colour inversion similar to the training data to evaluate MetaMorph. We then loaded the final trained model from the previous chapter and predicted the category and bounding boxes of the preprocessed LoFi sketches. We used the Python implementation of COCO detection metrics to calculate the final metrics with the ground-truth LoFi sketch annotations and predicted bounding boxes.

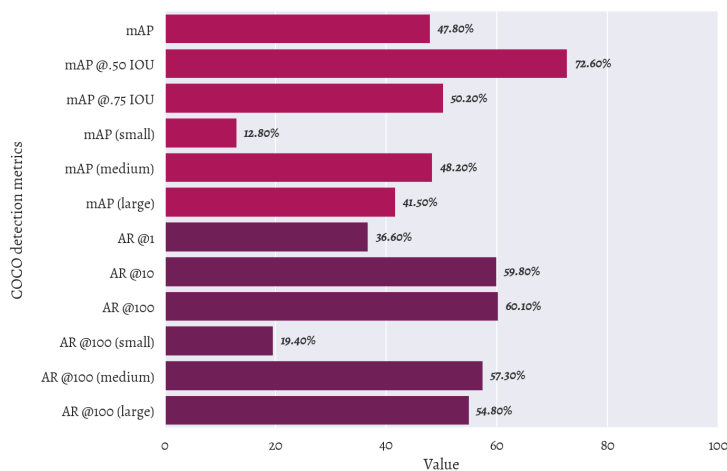
## 25.4 RESULTS

MetaMorph provides 47.8% mean Average Precision(mAP) and 36.6% Average Recall (AR) on this LoFi sketch evaluation dataset. The mAP for 0.5 IoU is the highest with 72.6%. The model provides near average mAP for medium sized (48.2%) and large (41.5%) UI element sketches. Similar to mAP the model provides adequate AR for medium (57.3%) and large (54.8%) UI element sketches. However, MetaMorph performs poorly for small sized UI element sketches (12.8% mAP and 19.4% AR). Table 25.2 and Figure 25.1 shows the COCO detection metrics of the MetaMorph.

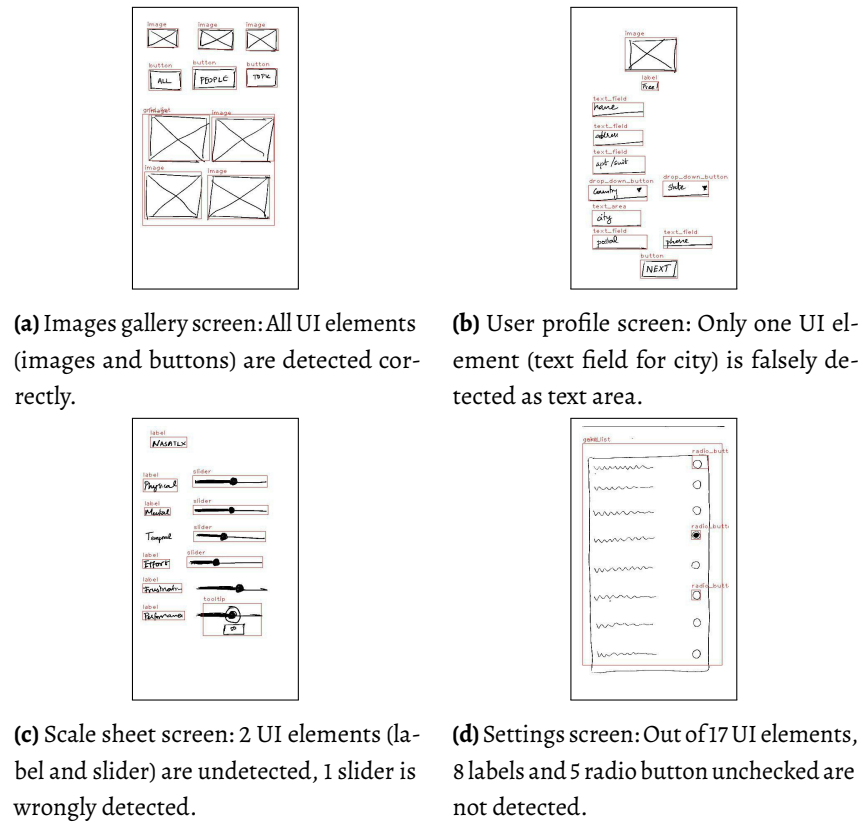
*MetaMorph provides adequate object detection for medium and large objects; however, it performs poorly for small-sized UI element sketches*

**Table 25.2:** COCO detection metrics of MetaMorph object detection model measured with the hand-drawn LoFi sketch evaluation dataset

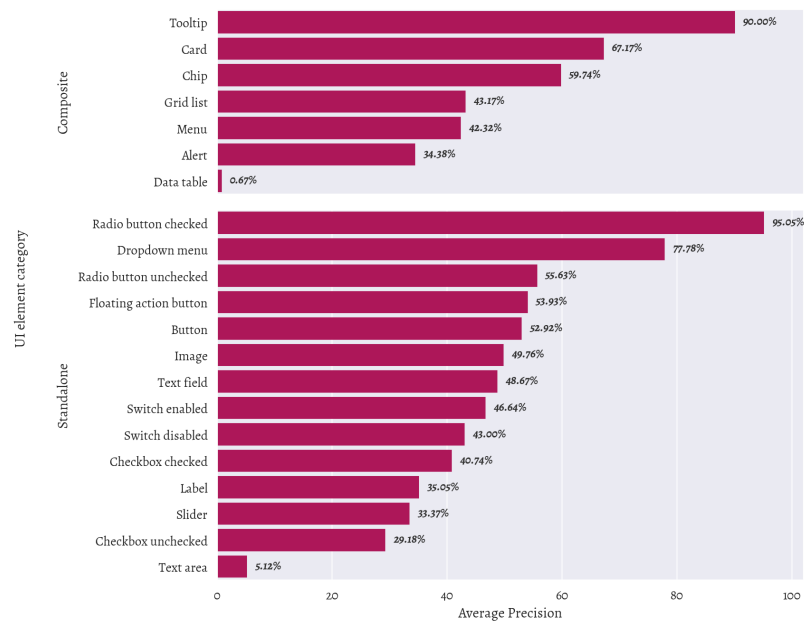
COCO Detection Metrics	Evaluation Result
mAP	47.8
mAP @.50 IOU	72.6
mAP @.75 IOU	50.2
mAP (small)	12.8
mAP (medium)	48.2
mAP (large)	41.5
AR @1	36.6
AR @10	59.8
AR @100	60.1
AR @100 (small)	19.4
AR @100 (medium)	57.3
AR @100 (large)	54.8



**Figure 25.1:** COCO detection metrics of MetaMorph UI element detector measured with the hand-drawn LoFi sketch evaluation dataset



**Figure 25.2:** Examples of UI element detection by MetaMorph using hand-drawn LoFi sketches of the user profile, images gallery, scale sheet, and settings screen.



**Figure 25.3:** UI element category wise average precision of MetaMorph UI element detector



**Table 25.3:** COCO detection metrics of MetaMorph object detection model measured with the hand-drawn LoFi sketch evaluation dataset

Category	UI element category	Average Precision
Composite	Tooltip	90.00
	Card	67.17
	Chip	59.74
	Grid list	43.17
	Menu	42.32
	Alert	34.38
	Data table	0.67
Standalone	Radio button checked	95.05
	Dropdown menu	77.78
	Radio button unchecked	55.63
	Floating action button	53.93
	Button	52.92
	Image	49.76
	Text field	48.67
	Switch enabled	46.64
	Switch disabled	43.00
	Checkbox checked	40.74
	Label	35.05
	Slider	33.37
	Checkbox unchecked	29.18
	Text area	5.12

Upon analyzing the average precision of the MetaMorph model on UI element category wise, we identified that the model identifies tooltip (90%) and radio button checked (95.05%) with the highest mAP. It identifies Dropdown menu (77.78%), Card (67.17%), Chip (59.74%), radio button unchecked (55.63%), floating action button (53.93%) and button (52.92%) with above-average mAP. The model performs very poorly for the Text area (5.12%) and Data table (0.67%). [Table 25.3](#) and [Figure 25.3](#) shows UI element category wise average precision of MetaMorph model evaluation.

*MetaMorph detects tooltip and radio button checked with the highest precision but performs very poorly in detecting text area and data table*

## 25.5 DISCUSSION

From the evaluation results, we understood that the MetaMorph UI element detector performs adequately for UI element detection with near-average mAP (47.8%). It generalizes detection for various UI element categories. However, it performs poorly for the Text area and Data table UI

*As the distribution of UI element categories differs in real-life LoFi sketches, we observe that MetaMorph provides adequate detection for the commonly occurring UI element categories*

elements. A key observation from the evaluation results is that as the distribution of UI element categories differs ([Table 25.1](#)) in real-life LoFi sketches, we observe that MetaMorph provides adequate detection for the commonly occurring UI element categories such as Label (35.05%), Image (49.76%), Button (52.92%), Card (67.17%), and Text field (48.67%). On the other hand, for rare UI element sketches such as Data table (0.67%), MetaMorph provides unsatisfactory results.

To further understand the user satisfaction by using MetaMorph, we conducted a user evaluation study by creating MetaMorph as a Web API and coupling it with Eve, a prototyping workbench created by Suleri, Pandian, et al. ([2019](#)). We discuss the user evaluation and its results in the next chapter.

# 26

---

## USER EVALUATION

---

From the evaluation metrics, we understood that MetaMorph provides adequate precision and recall for detecting UI elements from LoFi sketches from the model evaluation. To further evaluate MetaMorph and understand user satisfaction, we conducted a quantitative and qualitative study. This chapter explains these study setup, participants, methodology, and result.

### 26.1 WEB API

A key focus of MetaMorph is to provide a modular and reusable interface to convert LoFi sketches to higher fidelities. To attain this, we created a Web API to providing a user-friendly abstraction and access to the MetaMorph UI element detector. We wrote this API in Python using PyTorch and FastAPI<sup>1</sup> frameworks.

*We created a web API for MetaMorph to create a modular and reusable interface*

This Web API<sup>2</sup> accepts two parameters in the HTTP request: the LoFi sketch as an image and the minimum detection threshold value (decimal between 0 to 1). The API calls the MetaMorph model and obtains the predicted constituent UI element categories and bounding boxes. Using the prediction result, the API responds with a list of UI element categories and their location, dimension, and the prediction certainty in JSON format.

### 26.2 EVE: PROTOTYPING WORKBENCH

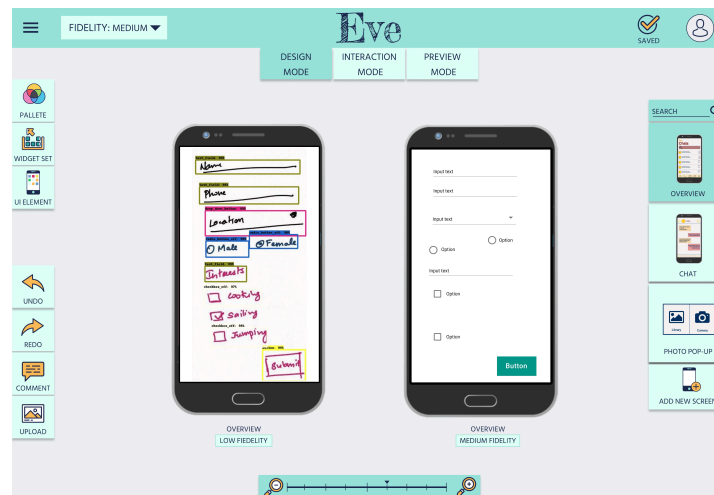
We used Eve: a prototyping workbench created by Suleri, Pandian, et al. (2019) to convert LoFi sketches to higher fidelities by coupling it with MetaMorph. Eve provides users with a canvas to sketch their UI designs as a LoFi prototype. This LoFi prototype is then passed to MetaMorph We-

*We coupled MetaMorph with Eve, a prototyping workbench for conducting user evaluation*

---

<sup>1</sup> <https://fastapi.tiangolo.com/>

<sup>2</sup> <https://api.metamorph.designwitheve.com/>



**Figure 26.1:** Eve: LoFi to MeFi transformation using UI element detection provided by MetaMorph (Pandian, Suleri, Beecks, et al., 2020)

bAPI, which then detects the sketched UI elements. Once the UI elements are detected, Eve then generates the respective UI elements as a MeFi prototype.

### 26.3 METHODOLOGY

*We used the standard  
After-Scenario  
Questionnaire (ASQ)  
for measuring user  
satisfaction in using  
MetaMorph*

To assess the designers' satisfaction in utilizing AI to transform LoFi to higher fidelities, we conducted a quantitative study using After-Scenario Questionnaire (ASQ), created by Lewis (1991b), followed by semi-structured interviews (15 min) to obtain qualitative feedback. The ASQ measures user satisfaction on the following three scales ease of completing the tasks (Q1) the amount of time it took to complete the tasks (Q2) the supporting information provided while completing the tasks (Q3).

#### 26.3.1 Participants

We used purposive and snowball sampling to recruit 10 UX designers (4 male and 6 female). The participants were between 23 to 35 years of age and had 2 - 8 years of prior prototyping experience.

#### 26.3.2 Scenario & Tasks

The participants were given a scenario to interact with Eve to create a LoFi prototype and transform it into a MeFi prototype utilizing UI element

detection provided by MetaMorph. They were instructed to complete the following five tasks.

1. Create a LoFi prototype by sketching 2 to 3 screens, e.g., login screen, user registration screen.
2. Use Eve to generate MeFi from the LoFi sketches
3. Identify correctly and wrongly identified UI elements
4. Correct the category of wrongly identified UI elements
5. Identify an undetected UI element

We also chose simple screens, such as a login screen with 3-5 UI element categories, and more complex screens, such as a user registration screen with around 5-10 UI element categories, to evaluate the MetaMorph model with different complexities. Also, keeping the flow of prototyping fidelity transformation in mind, we always kept the first two tasks at the beginning of the study. The order of the remaining tasks was randomized.

### 26.3.3 *Material & Apparatus*

Participants performed the aforementioned tasks in a lab setup. We provided each participant with a Microsoft Surface Studio and a stylus for sketching the LoFi prototypes.

### 26.3.4 *Procedure*

Participants were invited to the lab and briefed about the study's purpose and the assigned task. Participants were given time to explore various features of Eve and MetaMorph. After this brief exploration period, participants performed the assigned tasks and provided feedback using a think-aloud protocol. Once the participant completed the tasks, they were asked to fill the ASQ. On average, participants took around 25 minutes to complete the tasks and fill the questionnaire. After the study, we conducted semi-structured interviews (approx. 15 mins) to understand the participant's view on the AI-assisted prototyping approach and better understand and interpret the ASQ results.

*Study was conducted  
in lab setting*

**Table 26.1:** The number of UI elements the participants sketched, the number of elements MetaMorph correctly identified, wrongly identified and the number of unidentified UI elements during qualitative study using Eve along with their ASQ results

ID	No. of Screens	Count of UI Elements	Correct	Undetected	Wrong	UI Element Detection		ASQ		
						Precision	Recall	Q1	Q2	Q3
P1	2	21	17	3	5	0.77	0.85	5	6	5
P2	2	19	11	3	6	0.65	0.79	3	5	4
P3	3	20	14	2	4	0.78	0.88	5	4	5
P4	2	29	18	5	10	0.64	0.78	5	4	4
P5	3	32	27	3	6	0.82	0.90	6	7	5
P6	2	15	8	2	7	0.53	0.80	5	5	6
P7	3	26	18	3	8	0.69	0.86	6	5	4
P8	2	15	9	5	3	0.75	0.64	4	6	6
P9	3	21	15	2	7	0.68	0.88	5	6	7
P10	2	12	9	2	3	0.75	0.82	5	5	7
Mean						0.71	0.82	4.9	5.3	5.3

## 26.4 RESULTS & DISCUSSION

*MetaMorph received an above-average satisfaction level for all three questions in the ASQ*

MetaMorph received an above-average satisfaction level for all three questions in the ASQ: ease of completing the tasks (4.9), the amount of time it took to complete the tasks (5.3), and the supporting information provided when completing the tasks (5.3).

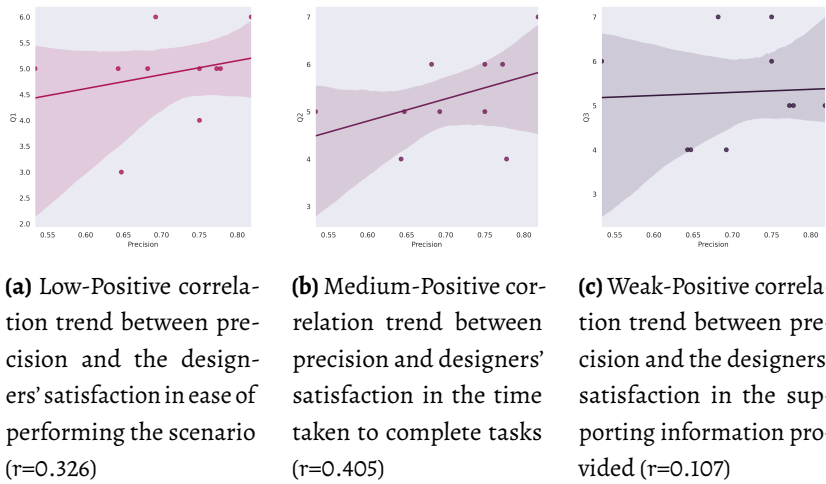
During this quantitative study, we also recorded the number of elements the participants sketched per screen, the number of UI elements MetaMorph correctly identified, wrongly identified, and the number of UI elements that remained undetected. As per the data collected during this study, MetaMorph provides 71% mAP and 82% AR (Table 26.1).

*We found that the user satisfaction of participants were positively correlated with the precision of MetaMorph's UI element detection.*

We further analyzed the correlation between the precision of UI element detection and designers' satisfaction. Unlike the ordinal values of the 7-point Likert scale in ASQ, the UI element detection precision is on an interval scale. Therefore, we converted the precision to an ordinal scale using average ranking and used rank correlation coefficient by Spearman (1904) to measure the correlation between them.

From this analysis, we discovered that there is a low-positive correlation between the precision of the UI element detector and the designers' satisfaction in ease of performing the scenario ( $r_{\text{precision} \times Q1} = 0.326$ ). There is a medium-positive correlation between precision and the designers' satisfaction in the time taken to complete tasks in the scenario

( $r_{\text{precision} \times Q2}=0.405$ ). However, only a weak-positive correlation was found ( $r_{\text{precision} \times Q3}=0.107$ ) between precision and the designers' satisfaction with the supporting information provided. Figure 26.2 visualizes the correlation trend between precision and the three questions from ASQ.



**Figure 26.2:** Spearman rank correlation between UI element detection precision and the three questions from ASQ

Based on the qualitative feedback provided by our participants, we understood that when the AI detects the UI elements precisely, it reduces their effort by providing them with a starting point. As a result, there is an increase in the ease of performing the scenario in relatively less time.

“ This system is simple and easy to use. This is less time consuming than what I usually do.

-P1

“ The result is almost accurate, now all I have to do is to remove the other stuff that I didn't want. I like that fact I don't have to adjust the size of the items (UI elements). But it would be ideal if the labels were also detected.

-P2

“ It (AI assistance) made it way quicker, and I had to do way less. So I could actually spend more time on polishing the design.

-P5

Based on the qualitative feedback, utilizing AI as an assistant at the end of LoFi prototyping was perceived as a novel and useful approach to

transforming LoFi sketches to higher fidelities. Participants expressed a willingness to use such an approach. Since MetaMorph only detects UI elements, it was suggested that in addition to detecting UI elements, text detection would add to the usefulness of the system. We aim to include this suggestion as part of our future work; we summarize MetaMorph, describe the limitations we identified and future work we plan in the next chapter.



---

## SUMMARY & FUTURE WORK

---

In this part of our thesis, we systematically researched utilising AI to assist at the end of the LoFi prototyping process by enabling the transformation of LoFi sketches to higher fidelities. To provide this assistance, we introduced MetaMorph, a DNN based UI element detector. To train MetaMorph, we used 300,377 synthetic LoFi sketches from the Syn and SynZ dataset and further fine-tuned it with 3,527 hand-drawn LoFi sketches from the LoFi sketch dataset. MetaMorph can detect 21 categories of UI elements from a given LoFi sketch. Upon evaluation with 895 real-life LoFi sketches, MetaMorph provides 47.8% mAP.

Further, to assess the designers' satisfaction in utilising AI at the end of the LoFi prototyping process to transform LoFi sketches to higher fidelities, we conducted a quantitative study using After-Scenario Questionnaire (ASQ) followed by semi-structured interviews to gather qualitative feedback. The results indicated that designers experience an above-average satisfaction level towards ease of task completion (4.9), time taken (5.3), and supporting information (5.3) upon utilising AI assistance for transforming LoFi sketches to higher fidelities. Their qualitative feedback indicated that they perceive utilising AI as a novel and useful approach during UI prototyping.

MetaMorph addresses the final part of RQ 2, 3, and 4 of this thesis. By creating and evaluating MetaMorph, this research is the first attempt to systematically research AI to transform LoFi sketches into higher fidelities. Through this research, we address the identified research gaps by training a reusable and generalisable AI model. We also measure the designer satisfaction in utilising AI assistance after LoFi prototyping to convert them to higher fidelities.

However, from the feedback of our user evaluation, we identified the following limitations with our current system, which we would like to fix in our future work.

From the qualitative feedback of the user evaluation, few participants requested to detect the text in the LoFi sketches. We further plan to couple MetaMorph with hand-writing recognition DNNs to add this feature to our system.

Overall, MetaMorph detects UI element sketches from a given LoFi sketch with adequate precision (47.8%). However, it performs poorly for small UI elements (12.8% mAP). We would like to investigate the reason further and improve its detection precision for small-sized UI elements. Also, MetaMorph performs poorly for few UI element categories such as Data table, Text area, and Checkbox unchecked. A potential reason could be the class imbalance between UI element categories in the LoFi sketch dataset. We would like to improve this in the next version.

As object detection research is continually developing, we would also like to experiment, improve, and update the underlying DNN for UI element detection in MetaMorph.

---

## CONCLUSION & FUTURE WORK

---

BlackBox toolkit results from our systematic research on providing AI assistance to UI designers before, during, and after the traditional LoFi prototyping process. This dissertation contributed four large-scale open-access datasets: UISketch, Syn & SynZ, LoFi Sketch and Wired datasets. These datasets were collected and refined from a diverse collection of data spanning participants from different countries, experience in prototyping, and input medium. Beyond the scope of this thesis, these datasets are readily available for training and evaluating various AI tools targeting LoFi, MeFi and HiFi prototypes. Further, through their diverse nature, they ensure the quality and generalizability of the AI models trained and evaluated with such datasets. Additionally, this research contributed three open-source tools: Akin, RUI TE, and MetaMorph.

Akin is a UI wireframe generator that assists UI designers **before LoFi prototyping** by generating multiple UI wireframes for a given UI design pattern; thus, enabling designers to conceptualise UI designs quickly. It used a modified conditional SAGAN for wireframe generation and provided an Inception Score of 1.63 (SD=0.34) and a Fréchet Inception Distance of 297.19. Our user evaluation revealed that UI/UX designers considered wireframes generated by Akin to be as good as wireframes made by designers. Moreover, designers identified Akin-generated wireframes as designer-made 50% of the time.

RUI TE is a UI wireframe refiner that assists UI designers **during LoFi prototyping** by aligning and grouping UI elements in a given UI wireframe. RUI TE used a Transformer-Encoder model and provided satisfactory results on almost all evaluation metrics: alignment score (0.87), improvement margin (~38%), grouping accuracy (20%), and mAP (58.53%). The qualitative feedback indicated that both novice and experienced participants preferred refinement of UI wireframe using AI in a button-click.

MetaMorph is a UI element detector that assists UI designers *after LoFi prototyping* by detecting the constituent UI elements of LoFi sketches and their location and dimension; thus, enabling the transformation of LoFi sketches to higher-fidelities. MetaMorph uses the RetinaNet object detection model and provides 47.8% mAP for hand-drawn LoFi sketches. The qualitative feedback showed that MetaMorph reduced their effort in transforming prototype fidelities and providing them with a starting point.

Overall, the UI designers perceived utilising AI for UI design as a novel and helpful approach and expressed their willingness to adopt it. Their ASQ results showed an above-average satisfaction level upon using all three AI assistance tools.

We can understand the impact of using the Blackbox toolkit by looking into how it assists in automating a few manual practices of LoFi prototyping. UI designers typically start their prototyping process by searching for inspirational UI designs then sketching or recreating them in a prototyping tool. They then modify the LoFi prototypes to fit their use case while either disregarding the aesthetics in LoFi sketches or carefully aligning the LoFi wireframes in the prototyping tool. However, UI designers who use the Blackbox toolkit for prototyping can utilise Akin to generate various editable LoFi wireframes for their particular use case before starting the prototyping process. While they edit the LoFi wireframe to fit their needs, they can utilise RUIE to align the UI wireframes with a click of a button. Further, they can also upload their LoFi sketches to Adobe XD prototyping platform and convert their sketches to LoFi wireframes or MeFi using the MetaMorph plugin instead of manually recreating their LoFi sketch to a higher fidelity prototype.

This thesis acts as a base for further research in adding AI assistance in UI prototyping. It introduces the approach to provide AI assistance to UI designers during the LoFi design process without sacrificing their autonomy. We believe this approach would greatly benefit designers where AI and humans co-create creative solutions. Besides, by the quantitative and qualitative user evaluation, this thesis provided a baseline metric on the performance of AI tools and their satisfaction upon using them before, during, and after LoFi prototyping. Moreover, these open-access datasets and open-source AI tools contributed through this thesis are not limited to the scope of the thesis and can be used in further UI design research in various applications due to their modular nature.

Despite these benefits, through the user evaluation, we identified several aspects in which the datasets and AI tools in the BlackBox toolkit could be improved. In general, the research on AI systems is improving tremendously as updated DNN architecture and parameter tuning methodologies are published every year. Therefore, the performance of the AI tools can be further fine-tuned or replaced by modifying the underlying DNN architecture of each AI tool. Moreover, the trends and aesthetics in UI design change drastically; therefore, there is a need to update the UI screens and UI components in the datasets and retrain the AI models in the future.

From a broader perspective, this thesis can further be expanded for various platforms such as desktop and web applications as it is currently scoped for smartphone applications. Also, as the datasets are not limited to the scope of this thesis, they open up many avenues of creating various AI tools supporting UI designers such as LoFi wireframe autocompletion, LoFi sketch-based UI screenshot retrieval, automated analysis of accessibility issues in UI screens, and adaptive UI by contextual auto-generation of UI screens.



Part VI

APPENDIX





# a

---

## COMPREHENSIVE LIST OF LITERATURE

---

This appendix chapter lists a comprehensive list of research and commercial projects that provide AI support to the UI design process with different research focuses. For each research project, we list the prototyping fidelity, dataset, and model it uses along with its model and user evaluation results.

**Table a.1:** List of literature that provide AI support in UI design by automating the prototype fidelity transformation

Research Work	Prototype	Dataset	Model / Algorithm	Model Evaluation	User Evaluation
Landy, 1996	LoFi - Sketch of Desktop apps	Not applicable	Rubine recognizer	-	-
Caetano et al., 2002	LoFi - Sketch of Desktop apps	Not applicable	Visual grammar using CALI recognizer	-	With 6 participants. Non-standard questionnaire. Users considered SketchIt as easy to use and to learn
Plimmer et al., 2003	LoFi - Sketch of Desktop apps	Not applicable	Rubine recognizer	-	With 16 students. Non-standard questionnaire. It increased student motivation on coding tasks significantly
Coyette et al., 2004	LoFi - Sketch of Desktop apps	Not applicable	CALI recognizer	-	-
Segura et al., 2012	LoFi - Sketch of Web apps	Not applicable	Based on Levenshtein distance, Douglas-Peucker algorithms	-	12 participants. Non-standard questionnaire. Participants had an overall positive opinion about UISKEI and its support on prototyping
Halbe et al., 2015	LoFi - Wireframe of Smartphone apps	Not specified	Learning Vector Quantization Neural Network	70% accuracy with 30 sketches	-
R. Huang et al., 2016	MeFi of Web apps	-	Line Detection heuristics and Random Forest	With 30 GUI screenshots. 0.782 Precision, 0.782 recall, 0.775 F1 score and 0.844 accuracy	-
Benjamin, 2017	LoFi - Sketch of Web apps	-	-	-	-
Beltramelli, 2017	MeFi of Smartphone, Web apps	5250 synthetic UI screenshots	GNN with LSTM (for code generation)	77% accuracy	-
S.-H. Li et al., 2017	LoFi - Sketch of Smartphone apps	Not applicable	Symbol recognition. Unspecified algorithm	-	Unspecified participants. Informal interview. Designers felt that creating UI components by sketching a line or rectangle is rapid and intuitive

*Table is continued in next page...*

Table a.1 continued from previous page

Research Work	Prototype	Dataset	Model/Algorithm	Model Evaluation	User Evaluation
Han et al., 2018	LoFi - Sketch of Web apps	1500 synthetic UI sketches	CNN, Bi-LSTM (for code generation) and SelfAttention layers	67.9% BLEU score	-
Kim et al., 2018	LoFi - Sketch of Web apps	-	Faster R-CNN	91% Precision and 86% recall	-
Microsoft AI Labs, 2018	LoFi - Sketch of Web apps	149 UI sketches, 8 categories of UI elements	Microsoft Custom Vision API	-	-
Park et al., 2018	LoFi - Sketch of Smartphone apps	-	YOLO for detecting object markers	-	-
Ashwin, 2018	LoFi - Sketch of Web apps	1750 website screenshots	VGG16 with GRU (for code generation)	76% BLEU score	-
C. Chen et al., 2018	MeFi of Smartphone apps	185277 UI screenshots with 291 categories of UI elements	CNN with RNN encoder-decoder	60.28% accuracy and 79.09 BLEU score	With 8 researchers. Non-standard questionnaire. Found that experiment group implements the skeleton GUIs faster than the control group.
Y. Liu et al., 2018	MeFi of Smartphone apps	5250 UI screenshots	CNN with BiLSTM (for code generation)	Accuracy fluctuating between 74% and 80%	-
Wallner, 2018	MeFi of Smartphone apps	1500 synthetic UI screenshots	CNN with LSTM (for code generation)	97% BLEU score.	-
Moran et al., 2018	MeFi of Smartphone apps	14,382 UI screenshots with 34 categories of UI elements	Canny edge detection and CNN based classifier	With 24,382 GUI images. 91.1% Precision	With 3 UI designers.
Bajammal et al., 2018	MeFi of Web apps	-	unsupervised visual matching using variable-density clustering	-	5 developers. : Comparison of automatically identified components to manually-identified ones by developers. System achieves an average of 94% precision and 75% recall in terms of agreement with the developers' assessment.
Jain et al., 2019	LoFi - Sketch of Web apps	150 UI sketches with 10 categories of UI elements	SSD RetinaNet	-	-

Table is continued in next page...

Table a.1 continued from previous page

Research Work	Prototype	Dataset	Model / Algorithm	Model Evaluation	User Evaluation
Yun et al., 2019	LoFi - Sketch of Web apps	50 UI sketches	YOLO	-	-
Uizard, 2019	LoFi - Sketch of Smartphone apps	Not specified	Not specified	Not specified	Not specified
Aşiroğlu et al., 2019	LoFi - Sketch of Web apps	Microsoft AI lab dataset + extra (186 UI sketches, 4 categories of UI elements)	Contour detection with CNN and BiLSTM for object classification	73% validation accuracy	-
S. Chen et al., 2019	MeFi of Smartphone apps	1842580 UI screenshots	Canny edge detection and CNN based classifier	85% accuracy in classification	-
Narendra et al., 2019	LoFi - Sketch of Web apps	Not applicable	Convex Hull detection algorithm	With 20 sketches. 100% accuracy for 5 UI elements and above 85% accuracy for other 3 UI elements	-
Sharma et al., 2020	MeFi of Web apps	1500 web screenshots	CNN with LSTM (for code generation)	With 250 images. 92% BLEU score	-
Mistry et al., 2020	LoFi - Sketch of Web apps	-	Contour detection with CNN for object classification	-	-
Wimmer et al., 2020	LoFi - Sketch of Web apps	Not applicable	DoodleClassifier	-	With 6 participants. Non-standard questionnaire. Participants were largely able to solve the given tasks and were satisfied with the results

**Table a.2:** List of literature that provide AI support in UI design by generating and refining LoFi UI wireframes or MeFi prototypes

Research Work	Prototype	Dataset	Model / Algorithm	Model Evaluation	User Evaluation
(Dayama et al., 2020)	MeFi of Web apps	-	Mixed Integer Linear Programming	-	With 16 designers 71.79 in System Usability Scale questionnaire and 53.04 in Creativity Support Index questionnaire
(Lee et al., 2020)	LoFi - Wireframe of Smartphone apps	RICO dataset (21000 UI screenshots with 13 categories of UI elements)	Neural Design Network which uses a Graph Neural Network and a conditional Variational Auto-Encoder (VAE)	FID 143.51±22.36	-
(J. Li et al., 2019)	LoFi - Wireframe of Smartphone apps	RICO dataset (Dataset details not specified)	Modified GAN with wireframe rendering discriminator	-	-
(Gupta et al., 2020)	LoFi - Wireframe of Smartphone apps	RICO dataset (62951 UI screenshots with 25 categories of UI elements)	Transformers	Spatial distribution analysis with 33.6 coverage and 23.7 overlap	-
(Zhao et al., 2021)	MeFi of Smartphone apps	RICO dataset (12688 UI screenshots with 25 categories of UI elements)	Modified SeqGAN	Average 0.075 FID and 0.869 1-NNA	5 grad students with non-standard questionnaire

**Table a.3:** List of literature that provide AI support in UI design by searching similar UI screenshots of given LoFi sketch or MeFi prototype

Research Work	Prototype	Dataset	Model / Algorithm	Model Evaluation	User Evaluation
Ge, 2019	LoFi - Sketch of Smartphone apps	RICO dataset (72000 UI screenshots with 7 categories of UI elements)	Similar to chenUIDesignImage2018	Not specified	-
F. Huang et al., 2019	LoFi - Sketch of Smartphone apps	3802 sketches	Modified VGG-A network	15.9% accuracy	With 5 designers. Qualitative feedback. Designers were satisfied and found the method useful.
Manandhar et al., 2020	MeFi of Smartphone apps	RICO dataset (66000 UI screenshots with 25 categories of UI elements) and GoogleUI dataset (18500 UI screenshots with 25 categories of UI elements)	Hybrid Encoder-Decoder backbone comprising a Graph Convolutional Network (GCN) and Convolutional Decoder (CNN).	61.7% Mean Intersection over Union, 70.1% Mean Pixel Accuracy	-

**Table a.4:** List of literature that provide AI support in UI design by autocompleting MeFi prototype during MeFi design process

Research Work	Prototype	Dataset	Model / Algorithm	Model Evaluation	User Evaluation
Y. Li et al., 2020	MeFi of Smartphone apps	RICO dataset (55000 UI screenshots with 25 categories of UI elements)	Transformer-Based Tree Decoders	F1 score 15.23% for retrieving parent-child pairs, next item accuracy 18.4%, and Edit distances 71.26	-

---

SEMANTIC ANNOTATIONS OF UI SCREENSHOTS

---

**Table b.1:** Exhaustive list of semantic UI element categories and its equivalent UISketch UI element category

Semantic UI Element Category	UISketch UI Element Category
About	Button
Ad	-
Address	Text Field
Age	Text Field
Age Picker	Dropdown
Amazon Button	Button
Available	-
Bottom Bar	Bottom Bar
Box Outline	-
Button	Button
Buy	Button
Captcha	-
Card	Card
Card Partial	-
Cart Add	Button
Cart Icon Button	Button
Check Box	Checkbox
City	Text Field
Color Picker	-
Continue	Button
Country	Text Field
Date Picker	Text Field
Discount	Label
Drop Down List	Dropdown
Edit Text	Text Field
Email	Text Field
Email Button	Button
Fb Button	Button

*Table is continued in next page...*

Table b.1 continued from previous page

Semantic UI element category	UISketch UI element category
Filter Button	Button
Forgot Password	Label
Gender Drop Down	Dropdown
Gender Picker	Dropdown
Google Button	Button
Heart Icon Button	Image
Icon Button	Button
Id	Text Field
Image Button	Button
Image View	Image
Instagram Button	Button
Last Name	Text Field
Linkedin	Button
Location	Text Field
Log In	Button
Map	-
Menu Dashes	Button
Menu Dots	Button
Menu Dots Dashes	Button
Message Icon	Button
Name	Text Field
Navigation Bar	Bottom Bar
Navigation Dots	-
Next	Button
Number	Text Field
Password	Text Field
Phone Button	Button
Phone Number	Text Field
Pin	Text Field
Price	Label
Price String	Label
Promo Code	Text Field
Quantity Counter	Dropdown
Radio Button	Radio Button
Rate It	Image
Rating Bar	Rating
Search Bar	Search Bar
Search Button	Button
Security Answer	-
Security Question	-
Select Button	Dropdown

*Table is continued in next page...*



Table b.1 continued from previous page

Semantic UI element category	UISketch UI element category
Separator Line	-
Separator Line Vertical	-
Settings Icon	Button
Share Button	Button
Sign Up	Button
Size Picker	Dropdown
Sm Button	Button
Snapchat Button	Button
Sort	Button
Sort Button	Button
Sort Selector	-
Star Icon Button	Button
Tab Bar	Tab Bar
Tab Bar Button	Button
Text Button Forgot Password	Label
Text Button Forgot Username	Label
Text Button Log In	Label
Text Button Privacy Terms	Label
Text Button Sign Up	Label
Text Button	Button
Text View	Label
Time	Text Field
Toggle	Switch
Top Bar	Top Bar
Twitter Button	Button
Username	Text Field
Whatsapp Button	Button
Windows Button	Button
Wish List	Button
Yahoo Button	Button
Zip Code	Text Field



- Abadi, Martín et al. (2015). “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.” In: <http://tensorflow.org/>.
- Albuquerque, Georgia, Thomas Löwe, and Marcus Magnor (Dec. 2011). “Synthetic Generation of High-Dimensional Datasets.” In: *IEEE transactions on visualization and computer graphics* 17, pp. 2317–24. DOI: [10.1109/tvcg.2011.237](https://doi.org/10.1109/tvcg.2011.237).
- Arjovsky, Martin, Soumith Chintala, and Léon Bottou (Dec. 2017). “Wasserstein GAN.” In: *arXiv:1701.07875 [cs, stat]*. <http://arxiv.org/abs/1701.07875>. arXiv: [1701.07875 \[cs, stat\]](https://arxiv.org/abs/1701.07875).
- Ashwin, Kumar (2018). *Sketch-Code*. GitHub. URL: <https://github.com/ashnkumar/sketch-code>.
- Aşıroğlu, Batuhan, Büşta Rümeysa Mete, Eyyüp Yıldız, Yağız Nalçakan, Alper Sezen, Mustafa Dağtekin, and Tolga Ensari (Apr. 2019). “Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques.” In: *2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT)*, pp. 1–4. DOI: [10.1109/EBBT.2019.8741736](https://doi.org/10.1109/EBBT.2019.8741736).
- Bajammal, Mohammad, Davood Mazinianian, and Ali Mesbah (Sept. 2018). “Generating Reusable Web Components from Mockups.” In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE 2018. New York, NY, USA: Association for Computing Machinery, pp. 601–611. ISBN: 978-1-4503-5937-5. DOI: [10.1145/3238147.3238194](https://doi.org/10.1145/3238147.3238194).
- Balsamiq (Feb. 2, 2021). Balsamiq. URL: <https://balsamiq.com/> (visited on 02/02/2021).
- Beltramelli, Tony (Sept. 2017). “Pix2code: Generating Code from a Graphical User Interface Screenshot.” In: *arXiv:1705.07962 [cs]*. arXiv: [1705.07962 \[cs\]](https://arxiv.org/abs/1705.07962).
- Benjamin, Wilkins (2017). *Sketching Interfaces - Generating Code from Low Fidelity Wireframes*. Airbnb Design.
- Borchers, Jan (2002). “Teaching HCI Design Patterns: Experience from Two University Courses.” In: *Patterns in Practice: A Workshop for UI Designers (at CHI 2002 International Conference on Human Factors of Computing Systems)*.

- Bradski, G. (2000). "The OpenCV Library." In: *Dr. Dobb's Journal of Software Tools*.
- Brown, Daniel M. (2011). *Communicating Design: Developing Web Site Documentation for Design and Planning*. en. 2nd ed. Berkeley, CA: New Riders. ISBN: 978-0-321-71246-2.
- Buxton, Bill (2011). *Sketching User Experiences: Getting the Design Right and the Right Design*. eng. Nachdr. Amsterdam: Morgan Kaufmann. ISBN: 978-0-12-374037-3.
- Caetano, Anabela, Neri Goulart, Manuel Fonseca, and Joaquim Jorge (2002). "JavaSketchIt: Issues in Sketching the Look of User Interfaces." In: *AAAI Spring Symposium on Sketch Understanding*. AAAI Press, Menlo Park, pp. 9–14.
- Chen, Chunyang, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu (May 2018). "From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation." In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE '18. Gothenburg, Sweden: Association for Computing Machinery, pp. 665–676. ISBN: 978-1-4503-5638-1. DOI: [10.1145/3180155.3180240](https://doi.org/10.1145/3180155.3180240).
- Chen, Sen, Lingling Fan, Ting Su, Lei Ma, Yang Liu, and Lihua Xu (Feb. 2019). "Automated Cross-Platform GUI Code Generation for Mobile Apps." In: *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, pp. 13–16. DOI: [10.1109/AI4Mobile.2019.8672718](https://doi.org/10.1109/AI4Mobile.2019.8672718).
- Coyette, Adrien, Stéphane Faulkner, Manuel Kolp, Quentin Limbourg, and Jean Vanderdonckt (2004). "SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML." In: *Proceedings of the 3rd Annual Conference on Task Models and Diagrams*. TAMODIA '04. New York, NY, USA: ACM, pp. 75–82. ISBN: 1-59593-000-0. DOI: [10.1145/1045446.1045461](https://doi.org/10.1145/1045446.1045461).
- Creswell, John W and J. David Creswell (2019). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*.
- Dayama, Niraj Ramesh, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta (Apr. 2020). "GRIDS: Interactive Layout Design with Integer Programming." In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI '20. New York, NY, USA: Association for Computing Machinery, pp. 1–13. ISBN: 978-1-4503-6708-0. DOI: [10.1145/3313831.3376553](https://doi.org/10.1145/3313831.3376553).
- Deka, Biplab, Zifeng Huang, Chad Franzen, Joshua Hibsichman, Daniel Afegan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar (Oct. 2017). "Rico: A Mobile App Dataset for Building Data-Driven Design Applications."

- In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. New York, NY, USA: Association for Computing Machinery, pp. 845–854. ISBN: 978-1-4503-4981-9. DOI: [10.1145/3126594.3126651](https://doi.org/10.1145/3126594.3126651).
- Denton, Emily, Soumith Chintala, Arthur Szlam, and Rob Fergus (June 2015). “Deep Generative Image Models Using a Laplacian Pyramid of Adversarial Networks.” In: *arXiv:1506.05751 [cs]*. <http://arxiv.org/abs/1506.05751>. arXiv: [1506.05751 \[cs\]](https://arxiv.org/abs/1506.05751).
- Dix, Alan, ed. (2004). *Human-Computer Interaction*. en. 3rd ed. Harlow, England ; New York: Pearson/Prentice-Hall. ISBN: 978-0-13-046109-4.
- Duyne, Douglas K. Van, James Landay, and Jason I. Hong (2002). *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-72149-X.
- Eitz, Mathias, James Hays, and Marc Alexa (July 2012). “How Do Humans Sketch Objects?” In: *ACM Transactions on Graphics* 31.4, 44:1–44:10. ISSN: 0730-0301. DOI: [10.1145/2185520.2185540](https://doi.org/10.1145/2185520.2185540).
- Engelberg, Daniel and Ahmed Seffah (2002). “A Framework for Rapid Mid-Fidelity Prototyping of Web Sites.” en. In: *Usability: Gaining a Competitive Edge*. Ed. by Judy Hammond, Tom Gross, and Janet Wesson. IFIP — The International Federation for Information Processing. Boston, MA: Springer US, pp. 203–215. ISBN: 978-0-387-35610-5. DOI: [10.1007/978-0-387-35610-5\\_14](https://doi.org/10.1007/978-0-387-35610-5_14).
- Erickson, Thomas (Aug. 2000). “*Lingua Francas* for Design: Sacred Places and Pattern Languages.” In: *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*. DIS '00. New York, NY, USA: Association for Computing Machinery, pp. 357–368. ISBN: 978-1-58113-219-9. DOI: [10.1145/347642.347794](https://doi.org/10.1145/347642.347794).
- Everingham, Mark, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman (June 2010). “The Pascal Visual Object Classes (VOC) Challenge.” In: *International Journal of Computer Vision* 88.2. <https://doi.org/10.1007/s11263-009-0275-4>, pp. 303–338. ISSN: 0920-5691. DOI: [10.1007/s11263-009-0275-4](https://doi.org/10.1007/s11263-009-0275-4).
- Fallman, Daniel (Apr. 5, 2003). “Design-Oriented Human-Computer Interaction.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: Association for Computing Machinery, pp. 225–232. ISBN: 978-1-58113-630-2. DOI: [10.1145/642611.642652](https://doi.org/10.1145/642611.642652). URL: <https://doi.org/10.1145/642611.642652> (visited on 04/16/2020).

- Fey, Matthias and Jan Eric Lenssen (Apr. 2019). "Fast Graph Representation Learning with PyTorch Geometric." In: *arXiv:1903.02428 [cs, stat]*. <http://arxiv.org/abs/1903.02428>. arXiv: 1903.02428 [cs, stat].
- Fonseca, Manuel J, César Pimentel, and Joaquim A Jorge (2002). "CALI: An Online Scribble Recognizer for Calligraphic Interfaces." In.
- Foster, David (2019). *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media.
- Framework7 (2020). *Framework7*. Framework7. URL: <https://framework7.io/> (visited on 09/11/2020).
- Gajjar, Nishit (Dec. 2020). "UI Generation Using Deep Learning from UI Design Patterns." en. MSc. Thesis. Aachen, Germany: RWTH Aachen University.
- Gajjar, Nishit, Vinoth Pandian Sermuga Pandian, Sarah Suleri, and Matthias Jarke (Mar. 2021). "Akin: Generating UI Wireframes from UI Design Patterns Using Deep Learning." In: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion*. IUI '21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3397482.3450727](https://doi.org/10.1145/3397482.3450727).
- Ge, Xiaofei (May 2019). "Android GUI Search Using Hand-Drawn Sketches." In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 141–143. DOI: [10.1109/ICSE-Companion.2019.00060](https://doi.org/10.1109/ICSE-Companion.2019.00060).
- GeekyAnts (Sept. 11, 2020). *NativeBase*. GeekyAnts. URL: <https://github.com/GeekyAnts/NativeBase> (visited on 09/11/2020).
- Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (Oct. 2014). "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." In: *arXiv:1311.2524 [cs]*. <http://arxiv.org/abs/1311.2524>. arXiv: 1311.2524 [cs].
- Girshick, Ross B. (2015). "Fast R-CNN." In: *CoRR abs/1504.08083*. arXiv: [1504.08083](https://arxiv.org/abs/1504.08083). URL: <http://arxiv.org/abs/1504.08083>.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.
- Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (June 10, 2014). *Generative Adversarial Networks*. arXiv: [1406.2661](https://arxiv.org/abs/1406.2661) [cs, stat]. URL: <http://arxiv.org/abs/1406.2661> (visited on 01/02/2021).
- Google (2021). *Material Design*. Material Design. URL: <https://material.io/>.

- Gupta, Kamal, Alessandro Achille, Justin Lazarow, Larry Davis, Vijay Mahadevan, and Abhinav Shrivastava (June 25, 2020). *Layout Generation and Completion with Self-Attention*. arXiv: 2006.14615 [cs]. URL: <http://arxiv.org/abs/2006.14615> (visited on 11/12/2020).
- Halbe, Aparna and Abhijit R. Joshi (2015). "A Novel Approach to HTML Page Creation Using Neural Network." en. In: *Procedia Computer Science* 45, pp. 197–204. ISSN: 18770509. DOI: 10.1016/j.procs.2015.03.122.
- Han, Yi, Jun He, and Qiwen Dong (Oct. 2018). "CSSSketch2Code: An Automatic Method to Generate Web Pages with CSS Style." In: *Proceedings of the 2nd International Conference on Advances in Artificial Intelligence*. ICAAI 2018. New York, NY, USA: Association for Computing Machinery, pp. 29–35. ISBN: 978-1-4503-6583-3. DOI: 10.1145/3292448.3292455.
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (Dec. 10, 2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs]. URL: <http://arxiv.org/abs/1512.03385> (visited on 09/15/2020).
- Herring, Scarlett R., Chia-Chen Chang, Jesse Krantzler, and Brian P. Bailey (Apr. 2009). "Getting Inspired! Understanding How and Why Examples Are Used in Creative Design Practice." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. New York, NY, USA: Association for Computing Machinery, pp. 87–96. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518717.
- Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (Jan. 2018). "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." In: *arXiv:1706.08500 [cs, stat]*. arXiv: 1706.08500 [cs, stat].
- Honda, Hiroto (July 2020). *Digging into Detectron 2*. en. <https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd>.
- Howard, Jeremy and Sylvain Gugger (July 2020a). *Deep Learning for Coders with Fastai and PyTorch*. Vol. 1. 1 vols. O'Reilly Media, Inc. 624 pp. ISBN: 978-1-4920-4552-6. URL: <https://www.oreilly.com/library/view/deep-learning-for/9781492045519/> (visited on 09/11/2020).
- Howard, Jeremy and Sylvain Gugger (2020b). "Fastai: A Layered API for Deep Learning." In: *Inf.* 11.2, p. 108. DOI: 10.3390/info11020108. URL: <https://doi.org/10.3390/info11020108>.
- Huang, Forrest, John F. Canny, and Jeffrey Nichols (May 2019). "Swire: Sketch-Based User Interface Retrieval." In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, pp. 1–10. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300334.

- Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger (Jan. 28, 2018). *Densely Connected Convolutional Networks*. arXiv: [1608.06993 \[cs\]](https://arxiv.org/abs/1608.06993). URL: <http://arxiv.org/abs/1608.06993> (visited on 09/15/2020).
- Huang, Jonathan et al. (2016). “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors.” In: *CoRR* abs/1611.10012. arXiv: [1611.10012](https://arxiv.org/abs/1611.10012). URL: <http://arxiv.org/abs/1611.10012>.
- Huang, Ruozhi, Yonghao Long, and Xiangping Chen (2016). *Automatically Generating Web Page From A Mockup*. en.
- Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer (Nov. 4, 2016). *SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size*. arXiv: [1602.07360 \[cs\]](https://arxiv.org/abs/1602.07360). URL: <http://arxiv.org/abs/1602.07360> (visited on 09/15/2020).
- Jain, Vanita, Piyush Agrawal, Subham Banga, Rishabh Kapoor, and Shashwat Gulyani (Oct. 2019). “Sketch2Code: Transformation of Sketches to UI in Real-Time Using Deep Neural Network.” In: *arXiv:1910.08930 [cs, eess]*. arXiv: [1910.08930 \[cs, eess\]](https://arxiv.org/abs/1910.08930).
- Kim, Bada, Sangmin Park, Taeyeon Won, Junyoung Heo, and Bongjae Kim (Oct. 2018). “Deep-Learning Based Web UI Automatic Programming.” In: *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*. RACS ’18. New York, NY, USA: Association for Computing Machinery, pp. 64–65. ISBN: 978-1-4503-5885-9. DOI: [10.1145/3264746.3264807](https://doi.org/10.1145/3264746.3264807).
- Kipf, Thomas N. and Max Welling (Feb. 2017). “Semi-Supervised Classification with Graph Convolutional Networks.” In: *arXiv:1609.02907 [cs, stat]*. <http://arxiv.org/abs/1609.02907>. arXiv: [1609.02907 \[cs, stat\]](https://arxiv.org/abs/1609.02907).
- Koch, Janin and Antti Oulasvirta (May 2016). “Computational Layout Perception Using Gestalt Laws.” In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, pp. 1423–1429. ISBN: 978-1-4503-4082-3.
- Krizhevsky, Alex (Apr. 26, 2014). *One Weird Trick for Parallelizing Convolutional Neural Networks*. arXiv: [1404.5997 \[cs\]](https://arxiv.org/abs/1404.5997). URL: <http://arxiv.org/abs/1404.5997> (visited on 09/15/2020).
- Landay, James A. (1996). “SILK: Sketching Interfaces Like Krazy.” In: *Conference Companion on Human Factors in Computing Systems*. CHI ’96. New



- York, NY, USA: ACM, pp. 398–399. ISBN: 0-89791-832-0. DOI: [10.1145/257089.257396](https://doi.org/10.1145/257089.257396).
- Landay, James A. and Brad A. Myers (1995). “Interactive Sketching for the Early Stages of User Interface Design.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '95. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., pp. 43–50. ISBN: 0-201-84705-1. DOI: [10.1145/223904.223910](https://doi.org/10.1145/223904.223910).
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (May 27, 2015). “Deep Learning.” In: *Nature* 521, p. 436. URL: <https://doi.org/10.1038/nature14539>.
- Lee, Hsin-Ying, Lu Jiang, Irfan Essa, Phuong B. Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang (July 2020). “Neural Design Network: Graphic Layout Generation with Constraints.” In: *arXiv:1912.09421 [cs]*. arXiv: [1912.09421 \[cs\]](https://arxiv.org/abs/1912.09421).
- Leiva, Luis A., Asutosh Hota, and Antti Oulasvirta (Oct. 2020). “Enrico: A Dataset for Topic Modeling of Mobile UI Designs.” In: *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI '20. New York, NY, USA: Association for Computing Machinery, pp. 1–4. ISBN: 978-1-4503-8052-2. DOI: [10.1145/3406324.3410710](https://doi.org/10.1145/3406324.3410710).
- Lewis, James R. (Jan. 1991a). “Psychometric Evaluation of an After-Scenario Questionnaire for Computer Usability Studies: The ASQ.” In: *ACM SIGCHI Bulletin* 23.1, pp. 78–81. ISSN: 0736-6906. DOI: [10.1145/122672.122692](https://doi.org/10.1145/122672.122692).
- Lewis, James R. (Jan. 1, 1991b). “Psychometric Evaluation of an After-Scenario Questionnaire for Computer Usability Studies: The ASQ.” In: *ACM SIGCHI Bulletin* 23.1, pp. 78–81. ISSN: 0736-6906. DOI: [10.1145/122672.122692](https://doi.org/10.1145/122672.122692). URL: <https://doi.org/10.1145/122672.122692> (visited on 08/13/2020).
- Li, Jianan, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu (Jan. 20, 2019). *LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators*. arXiv: [1901.06767 \[cs\]](https://arxiv.org/abs/1901.06767). URL: <http://arxiv.org/abs/1901.06767> (visited on 04/30/2020).
- Li, Shu-Hui, Jia-Jyun Hsu, Chih-Ya Chang, Pin-Hsuan Chen, and Neng-Hao Yu (June 2017). “Xketch: A Sketch-Based Prototyping Tool to Accelerate Mobile App Design Process.” In: *Proceedings of the 2017 ACM Conference Companion Publication on Designing Interactive Systems*. DIS'17 Companion. New York, NY, USA: Association for Computing Machinery, pp. 301–304. ISBN: 978-1-4503-4991-8. DOI: [10.1145/3064857.3079179](https://doi.org/10.1145/3064857.3079179).

- Li, Yang, Julien Amelot, Xin Zhou, Samy Bengio, and Si Si (Jan. 2020). “Auto Completion of User Interface Layout Design Using Transformer-Based Tree Decoders.” In: *arXiv:2001.05308 [cs]*. arXiv: [2001.05308 \[cs\]](https://arxiv.org/abs/2001.05308).
- Lin, Tsung Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick (2014). “Microsoft COCO: Common Objects in Context.” In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8693 Lncs. Part 5, pp. 740–755. DOI: [10.1007/978-3-319-10602-1\\_48](https://doi.org/10.1007/978-3-319-10602-1_48). URL: <http://cocodataset.org/#home>.
- Lin, Tsung-Yi, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár (2017). “Focal Loss for Dense Object Detection.” In: *CoRR abs/1708.02002*. arXiv: [1708.02002](https://arxiv.org/abs/1708.02002). URL: <http://arxiv.org/abs/1708.02002>.
- Liu, Thomas F., Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar (Oct. 2018). “Learning Design Semantics for Mobile Apps.” In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST’18. Berlin, Germany: Association for Computing Machinery, pp. 569–579. ISBN: 978-1-4503-5948-1. DOI: [10.1145/3242587.3242650](https://doi.org/10.1145/3242587.3242650).
- Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg (2016). “SSD: Single Shot MultiBox Detector.” In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, pp. 21–37. ISBN: 978-3-319-46448-0.
- Liu, Yanbin, Qidi Hu, and Kunxian Shu (Nov. 2018). “Improving Pix2code Based Bi-Directional LSTM.” In: *2018 IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, pp. 220–223. DOI: [10.1109/AUTEEE.2018.8720784](https://doi.org/10.1109/AUTEEE.2018.8720784).
- Manandhar, Dipu, Dan Ruta, and John Collomosse (2020). “Learning Structural Similarity of User Interface Layouts Using Graph Networks.” In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 730–746. ISBN: 978-3-030-58542-6. DOI: [10.1007/978-3-030-58542-6\\_44](https://doi.org/10.1007/978-3-030-58542-6_44).
- Microsoft (2021). *Microsoft Design*. Fluent Design System. URL: <https://www.microsoft.com/design/fluent/> (visited on 09/11/2020).
- Microsoft AI Labs (2018). *Sketch2Code - Microsoft/Ailab*. GitHub. URL: <https://github.com/Microsoft/ailab/tree/master/Sketch2Code>.

- Mirza, Mehdi and Simon Osindero (Nov. 2014). “Conditional Generative Adversarial Nets.” In: *arXiv:1411.1784 [cs, stat]*. <http://arxiv.org/abs/1411.1784>. arXiv: 1411.1784 [cs, stat].
- Mistry, Mihir, Ameya Apte, Varad Ghodake, and S. B. Mane (2020). “Machine Learning Based User Interface Generation.” en. In: *Intelligent Computing, Information and Control Systems*. Ed. by A. Pasumpon Pandian, Klimis Ntalianis, and Ram Palanisamy. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, pp. 453–460. ISBN: 978-3-030-30465-2. DOI: [10.1007/978-3-030-30465-2\\_50](https://doi.org/10.1007/978-3-030-30465-2_50).
- Moqups (Feb. 2, 2021). Moqups. URL: <https://moqups.com> (visited on 02/02/2021).
- Moran, Kevin, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk (June 2018). “Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps.” In: *arXiv:1802.02312 [cs]*. arXiv: [1802.02312 \[cs\]](https://arxiv.org/abs/1802.02312).
- Mostow, Jack (1985). “Toward Better Models of the Design Process.” In: *AI Mag.* 6.1, pp. 44–57.
- Narendra, Savinay, Sheelabhadra Dey, Josiah Coad, Seth Polsley, and Tracy Hammond (2019). “FreeStyle: A Sketch-Based Wireframing Tool.” en. In: *Inspiring Students with Digital Ink: Impact of Pen and Touch on Education*. Ed. by Tracy Hammond, Manoj Prasad, and Anna Stepanova. Human–Computer Interaction Series. Cham: Springer International Publishing, pp. 105–117. ISBN: 978-3-030-17398-2. DOI: [10.1007/978-3-030-17398-2\\_7](https://doi.org/10.1007/978-3-030-17398-2_7).
- NinjaMock (Feb. 2, 2021). NinjaMock. URL: <https://ninjamock.com> (visited on 02/02/2021).
- O’Donovan, Peter, Aseem Agarwala, and Aaron Hertzmann (Apr. 2015). “DesignScape: Design with Interactive Layout Suggestions.” In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI ’15. New York, NY, USA: Association for Computing Machinery, pp. 1221–1224. ISBN: 978-1-4503-3145-6. DOI: [10.1145/2702123.2702149](https://doi.org/10.1145/2702123.2702149).
- ODonovan, Peter, Aseem Agarwala, and Aaron Hertzmann (Aug. 2014). “Learning Layouts for Single-PageGraphic Designs.” In: *IEEE Transactions on Visualization and Computer Graphics* 20.8, pp. 1200–1213. ISSN: 1077-2626. DOI: [10.1109/TVCG.2014.48](https://doi.org/10.1109/TVCG.2014.48).
- Oord, Aaron van den, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu (June 2016). “Conditional Image

- Generation with PixelCNN Decoders.” In: *arXiv:1606.05328 [cs]*. <http://arxiv.org/abs/1606.05328>. arXiv: 1606.05328 [cs].
- Otsu, Nobuyuki (1979). “A Threshold Selection Method from Gray-Level Histograms.” In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66. DOI: [10.1109/TSMC.1979.4310076](https://doi.org/10.1109/TSMC.1979.4310076).
- Padilla, Rafael, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva (Jan. 2021). “A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit.” en. In: *Electronics* 10.3. <https://www.mdpi.com/2079-9292/10/3/279>, p. 279. ISSN: 2079-9292. DOI: [10.3390/electronics10030279](https://doi.org/10.3390/electronics10030279).
- Palmer, Stephen E (1999). *Vision Science: Photons to Phenomenology*. MIT press.
- Pandian, Vinoth Pandian Sermuga, Abdullah Shams, Sarah Suleri, and Matthias Jarke (2022). “LoFi Sketch: A Large Scale Dataset of Smartphone Low Fidelity Sketches.” In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA ’22. <http://doi.acm.org/10.1145/3491101.3519624>. New York, NY, USA: ACM. ISBN: 978-1-4503-9156-6/22/04. DOI: [10.1145/3491101.3519624](https://doi.org/10.1145/3491101.3519624).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, Christian Beecks, and Matthias Jarke (2020). “MetaMorph: AI Assistance to Transform Lo-Fi Sketches to Higher Fidelities.” In: *32nd Australian Conference on Human-Computer Interaction*. OzCHI ’20. <https://doi.org/10.1145/3441000.3441030>. New York, NY, USA: Association for Computing Machinery, pp. 403–412. ISBN: 978-1-4503-8975-4. DOI: [10.1145/3441000.3441030](https://doi.org/10.1145/3441000.3441030).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (Mar. 2020). “Syn: Synthetic Dataset for Training UI Element Detector From Lo-Fi Sketches.” In: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. IUI ’20. <https://doi.org/10.1145/3379336.3381498>. Cagliari, Italy: Association for Computing Machinery, pp. 79–80. ISBN: 978-1-4503-7513-9. DOI: [10.1145/3379336.3381498](https://doi.org/10.1145/3379336.3381498).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (Mar. 2021a). “SynZ: Enhanced Synthetic Dataset for Training UI Element Detectors.” In: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion*. IUI ’21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3397482.3450725](https://doi.org/10.1145/3397482.3450725).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (2021b). “UISketch: A Large-Scale Dataset of UI Element Sketches.” In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI

- '21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3411764.3445784](https://doi.org/10.1145/3411764.3445784).
- Park, Jisu, Young-Sun Yun, Seongbae Eun, Sin Cha, Sun-Sup So, and Jinman Jung (Oct. 2018). "Deep Neural Networks Based User Interface Detection for Mobile Applications Using Symbol Marker." In: *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*. RACS '18. Honolulu, Hawaii: Association for Computing Machinery, pp. 66–67. ISBN: 978-1-4503-5885-9. DOI: [10.1145/3264746.3264808](https://doi.org/10.1145/3264746.3264808).
- Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- Patki, Neha, Roy Wedge, and Kalyan Veeramachaneni (Oct. 2016). "The Synthetic Data Vault." en. In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Montreal, QC, Canada: IEEE, pp. 399–410. ISBN: 978-1-5090-5206-6. DOI: [10.1109/DSAA.2016.49](https://doi.org/10.1109/DSAA.2016.49).
- Perez, Medina and Luis Jorge (2016). "The UsiSketch Software Architecture." In: *Romanian Journal of Human-Computer Interaction* 9.4, pp. 305–333. URL: <https://dial.uclouvain.be/pr/boreal/object/boreal:187342>.
- Plimmer, Beryl and Mark D. Apperley (2003). "Software for Students to Sketch Interface Designs." In: *Human-Computer Interaction INTERACT '03: IFIP TC13 International Conference on Human-Computer Interaction, 1st-5th September 2003, Zurich, Switzerland*. IOS Press.
- Radford, Alec, Luke Metz, and Soumith Chintala (Jan. 2016). "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *arXiv:1511.06434 [cs]*. <http://arxiv.org/abs/1511.06434>. arXiv: [1511.06434 \[cs\]](https://arxiv.org/abs/1511.06434).
- Rahman, Soliha, Vinoth Pandian Sermuga Pandian, and Matthias Jarke (Mar. 2021). "RUIITE: Refining UI Layout Aesthetics Using Transformer Encoder." In: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion*. IUI '21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3397482.3450716](https://doi.org/10.1145/3397482.3450716).
- React Native Elements (Sept. 11, 2020). *React Native Elements*. React Native Elements. URL: <https://github.com/react-native-elements/react-native-elements> (visited on 09/11/2020).

- Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi (May 2016). "You Only Look Once: Unified, Real-Time Object Detection." In: *arXiv:1506.02640 [cs]*. <http://arxiv.org/abs/1506.02640>. arXiv: [1506.02640 \[cs\]](http://arxiv.org/abs/1506.02640).
- Ren, Shaoqing, Kaiming He, Ross B. Girshick, and Jian Sun (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *CoRR abs/1506.01497*. arXiv: [1506.01497](http://arxiv.org/abs/1506.01497). URL: <http://arxiv.org/abs/1506.01497>.
- Rubine, Dean (July 1991). "Specifying Gestures by Example." In: *SIGGRAPH Comput. Graph.* 25.4, pp. 329–337. ISSN: 0097-8930. DOI: [10.1145/127719.122753](https://doi.org/10.1145/127719.122753).
- Rudd, Jim, Ken Stern, and Scott Isensee (Jan. 1996). "Low vs. High-Fidelity Prototyping Debate." In: *Interactions* 3.1, pp. 76–85. ISSN: 1072-5520. DOI: [10.1145/223500.223514](https://doi.org/10.1145/223500.223514).
- Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (June 2016). "Improved Techniques for Training GANs." In: *arXiv:1606.03498 [cs]*. arXiv: [1606.03498 \[cs\]](http://arxiv.org/abs/1606.03498).
- Sangkloy, Patsorn, Nathan Burnell, Cusuh Ham, and James Hays (July 2016). "The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies." In: *ACM Transactions on Graphics* 35.4, 119:1–119:12. ISSN: 0730-0301. DOI: [10.1145/2897824.2925954](https://doi.org/10.1145/2897824.2925954).
- Seddati, Omar, Stéphane Dupont, and Saïd Mahmoudi (June 2015). "DeepSketch: Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search." In: *2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI), pp. 1–6. DOI: [10.1109/CBMI.2015.7153606](https://doi.org/10.1109/CBMI.2015.7153606).
- Seffah, Ahmed (Sept. 2003). "Learning the Ropes: Human-Centered Design Skills and Patterns for Software Engineers' Education." In: *Interactions* 10.5, pp. 36–45. ISSN: 1072-5520. DOI: [10.1145/889692.889693](https://doi.org/10.1145/889692.889693).
- Segura, Vinícius C. V. B., Simone D. J. Barbosa, and Fabiana Pedreira Simões (May 2012). "UISKEI: A Sketch-Based Prototyping Tool for Defining and Evaluating User Interface Behavior." In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI '12. New York, NY, USA: Association for Computing Machinery, pp. 18–25. ISBN: 978-1-4503-1287-5. DOI: [10.1145/2254556.2254564](https://doi.org/10.1145/2254556.2254564).
- Shams, Abdullah (Apr. 6, 2021). "UISketch: A Dataset Of Hand-Drawn Lo-Fi Sketches." MSc. Thesis. Aachen, Germany: RWTH Aachen University. 83 pp.



- Sharma, Prerna, Vikas Chaudhary, Nakul Malhotra, Nikita Gupta, and Mohit Mittal (2020). "Dynamic Web with Automatic Code Generation Using Deep Learning." en. In: *International Conference on Innovative Computing and Communications*. Ed. by Ashish Khanna, Deepak Gupta, Siddhartha Bhattacharyya, Vaclav Snasel, Jan Platos, and Aboul Ella Hassanien. Advances in Intelligent Systems and Computing. Singapore: Springer, pp. 687–697. ISBN: 9789811512865. DOI: [10.1007/978-981-15-1286-5\\_61](https://doi.org/10.1007/978-981-15-1286-5_61).
- Shneiderman, Ben (Feb. 2020a). "Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy." In: *arXiv:2002.04087 [cs]*. arXiv: [2002.04087 \[cs\]](https://arxiv.org/abs/2002.04087).
- Shneiderman, Ben (2020b). "Human-Centered Artificial Intelligence: Three Fresh Ideas." en. In: *AIS Transactions on Human-Computer Interaction*, pp. 109–124. ISSN: 19443900. DOI: [10.17705/1thci.00131](https://doi.org/10.17705/1thci.00131).
- Shneiderman, Ben and Catherine Plaisant (2004). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 4th ed. Boston: Pearson/Addison Wesley. 652 pp. ISBN: 978-0-321-19786-3.
- Shoutem (Sept. 10, 2020). *Shoutem UI*. Shoutem. URL: <https://github.com/shoutem/ui> (visited on 09/11/2020).
- Simonyan, Karen and Andrew Zisserman (Apr. 2015a). "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *arXiv:1409.1556 [cs]*. arXiv: [1409.1556 \[cs\]](https://arxiv.org/abs/1409.1556).
- Simonyan, Karen and Andrew Zisserman (Apr. 10, 2015b). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: [1409.1556 \[cs\]](https://arxiv.org/abs/1409.1556). URL: <http://arxiv.org/abs/1409.1556> (visited on 09/15/2020).
- Smith, Leslie N. (Apr. 4, 2017). *Cyclical Learning Rates for Training Neural Networks*. arXiv: [1506.01186 \[cs\]](https://arxiv.org/abs/1506.01186). URL: <http://arxiv.org/abs/1506.01186> (visited on 09/11/2020).
- Snyder, Carolyn (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. en. The Morgan Kaufmann Series in Interactive Technologies. San Diego, CA: Morgan Kaufmann Pub. ISBN: 978-1-55860-870-2.
- Spearman, C. (1904). "The Proof and Measurement of Association between Two Things." In: *The American Journal of Psychology* 15.1, pp. 72–101. ISSN: 00029556. JSTOR: [1412159](https://www.jstor.org/stable/1412159).
- Suleri, Sarah, Nilda Kipi, Linh Chi Tran, and Matthias Jarke (Oct. 2019). "UI Design Pattern-Driven Rapid Prototyping for Agile Development of Mobile Applications." In: *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI

- '19. <https://doi.org/10.1145/3338286.3344399>. New York, NY, USA: Association for Computing Machinery, pp. 1–6. ISBN: 978-1-4503-6825-4. DOI: [10.1145/3338286.3344399](https://doi.org/10.1145/3338286.3344399).
- Suleri, Sarah, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke (2019). “Eve: A Sketch-Based Software Prototyping Workbench.” In: *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK). CHI EA '19. New York, NY, USA: ACM, Lbw1410:1–lbw1410:6. ISBN: 978-1-4503-5971-9. DOI: [10.1145/3290607.3312994](https://doi.org/10.1145/3290607.3312994). URL: <http://doi.acm.org/10.1145/3290607.3312994>.
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (Sept. 16, 2014). *Going Deeper with Convolutions*. arXiv: [1409.4842](https://arxiv.org/abs/1409.4842) [cs]. URL: <http://arxiv.org/abs/1409.4842> (visited on 09/15/2020).
- Szegedy, Christian, Alexander Toshev, and Dumitru Erhan (Dec. 2013). “Deep Neural Networks for Object Detection.” In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., pp. 2553–2561.
- Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna (Dec. 11, 2015). *Rethinking the Inception Architecture for Computer Vision*. arXiv: [1512.00567](https://arxiv.org/abs/1512.00567) [cs]. URL: <http://arxiv.org/abs/1512.00567> (visited on 09/15/2020).
- Tan, Mingxing and Quoc V. Le (Dec. 1, 2019). *MixConv: Mixed Depthwise Convolutional Kernels*. arXiv: [1907.09595](https://arxiv.org/abs/1907.09595) [cs]. URL: <http://arxiv.org/abs/1907.09595> (visited on 09/15/2020).
- Tan, Mingxing and Quoc V. Le (Sept. 11, 2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Version 5. arXiv: [1905.11946](https://arxiv.org/abs/1905.11946) [cs, stat]. URL: <http://arxiv.org/abs/1905.11946> (visited on 09/15/2020).
- Tidwell, Jenifer, Charles Brewer, Aynne Valencia, and an O'Reilly Media Company Safari (2020). *Designing Interfaces, 3rd Edition*. English.
- Tony, Beltramelli (2017). “Teaching Machines to Understand User Interfaces.” In.
- Tractinsky, Noam, Avivit Cokhavi, Moti Kirschenbaum, and Tal Sharfi (Nov. 2006). “Evaluating the Consistency of Immediate Aesthetic Perceptions of Web Pages.” en. In: *International Journal of Human-Computer Studies* 64.11. <http://www.sciencedirect.com/science/article/pii/>



- S1071581906000863, pp. 1071–1083. ISSN: 1071-5819. DOI: [10.1016/j.ijhcs.2006.06.009](https://doi.org/10.1016/j.ijhcs.2006.06.009).
- Tuch, Alexandre N., Eva Presslauer, Markus Stoecklin, Klaus Opwis, and Javier Bargas-Avila (2012). “The Role of Visual Complexity and Prototypicality Regarding First Impression of Websites: Working towards Understanding Aesthetic Judgments.” In: *International Journal of Human-Computer Studies* 70(11). <http://dx.doi.org/10.1016/j.ijhcs.2012.06.003>, pp. 794–811.
- Uijlings, J. R. R., K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders (2013). “Selective Search for Object Recognition.” In: *International Journal of Computer Vision* 104.2, pp. 154–171. URL: <https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013>.
- Uizard (2019). *Uizard for Design*. en. <https://uizard.io/product>. Commercial.
- Van der Maaten, Laurens and Geoffrey Hinton (2008). “Visualizing Data Using T-SNE.” In: *Journal of Machine Learning Research* 9, pp. 2579–2605. URL: <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (Dec. 2017). “Attention Is All You Need.” In: *arXiv:1706.03762 [cs]*. <http://arxiv.org/abs/1706.03762>. arXiv: [1706.03762 \[cs\]](https://arxiv.org/abs/1706.03762).
- Wada, Kentaro (2016). *Labelme: Image Polygonal Annotation with Python*.
- Walker, Miriam, Leila Takayama, and James A. Landay (2002). “High-Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes When Testing Web Prototypes.” In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 46.5, pp. 661–665. URL: <https://doi.org/10.1177/154193120204600513>.
- Wallner, Emil (Jan. 2018). *Turning Design Mockups Into Code With Deep Learning*. en. <https://blog.floydhub.com/turning-design-mockups-into-code-with-deep-learning/>. Blog.
- Wilson, James and Daniel Rosenberg (1988). “Rapid Prototyping for User Interface Design.” en. In: Elsevier. ISBN: 978-0-444-70536-5. DOI: [10.1016/B978-0-444-70536-5.50044-0](https://doi.org/10.1016/B978-0-444-70536-5.50044-0).
- Wimmer, Christoph, Alex Untertrifaller, and Thomas Grechenig (Dec. 2020). “SketchingInterfaces: A Tool for Automatically Generating High-Fidelity User Interface Mockups from Hand-Drawn Sketches.” In: *32nd Australian Conference on Human-Computer Interaction*. OzCHI ’20. New York, NY, USA: Association for Computing Machinery, pp. 538–545. ISBN: 978-1-4503-8975-4. DOI: [10.1145/3441000.3441015](https://doi.org/10.1145/3441000.3441015).

- Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick (2019). "Detectron2." In: <https://github.com/facebookresearch/detectron2>.
- Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (Apr. 10, 2017). *Aggregated Residual Transformations for Deep Neural Networks*. arXiv: [1611.05431](https://arxiv.org/abs/1611.05431) [cs]. URL: <http://arxiv.org/abs/1611.05431> (visited on 09/15/2020).
- Yun, Young-Sun, Jinman Jung, Seongbae Eun, Sun-Sup So, and Junyoung Heo (2019). "Detection of GUI Elements on Sketch Images Using Object Detector Based on Deep Neural Networks." en. In: *Proceedings of the Sixth International Conference on Green and Human Information Technology*. Ed. by Seong Oun Hwang, Syh Yuan Tan, and Franklin Bien. Lecture Notes in Electrical Engineering. Singapore: Springer, pp. 86–90. ISBN: 9789811303111. DOI: [10.1007/978-981-13-0311-1\\_16](https://doi.org/10.1007/978-981-13-0311-1_16).
- Zagoruyko, Sergey and Nikos Komodakis (June 14, 2017). *Wide Residual Networks*. arXiv: [1605.07146](https://arxiv.org/abs/1605.07146) [cs]. URL: <http://arxiv.org/abs/1605.07146> (visited on 09/15/2020).
- Zhang, Han, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena (June 2019). "Self-Attention Generative Adversarial Networks." In: *arXiv:1805.08318* [cs, stat]. arXiv: [1805.08318](https://arxiv.org/abs/1805.08318) [cs, stat].
- Zhao, Tianming, Chunyang Chen, Yuanning Liu, and Xiaodong Zhu (Jan. 2021). "GUIGAN: Learning to Generate GUI Designs Using Generative Adversarial Networks." In: *arXiv:2101.09978* [cs]. arXiv: [2101.09978](https://arxiv.org/abs/2101.09978) [cs].
- Zhou, Jie, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun (Jan. 2020). "Graph Neural Networks: A Review of Methods and Applications." en. In: *AI Open* 1. <https://www.sciencedirect.com/science/article/pii/S2666651021000012>, pp. 57–81. ISSN: 2666-6510. DOI: [10.1016/j.aiopen.2021.01.001](https://doi.org/10.1016/j.aiopen.2021.01.001).

- Aragón, Gustavo, Erdem Gümrükcü, Vinoth Pandian Sermuga Pandian, and Otilia Werner-Kytölä (2019). "Cooperative Control of Charging Stations for an EV Park with Stochastic Dynamic Programming." In: *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*. Vol. 1. IEEE, pp. 6649–6654.
- Pandian, Vinoth Pandian Sermuga and Sarah Suleri (2020a). "BlackBox Toolkit: Intelligent Assistance to UI Design." In: *arXiv e-prints*, arXiv–2004.
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, Christian Beecks, and Matthias Jarke (2020). "MetaMorph: AI Assistance to Transform Lo-Fi Sketches to Higher Fidelities." In: *32nd Australian Conference on Human-Computer Interaction*. OzCHI '20. <https://doi.org/10.1145/3441000.3441030>. New York, NY, USA: Association for Computing Machinery, pp. 403–412. ISBN: 978-1-4503-8975-4. DOI: [10.1145/3441000.3441030](https://doi.org/10.1145/3441000.3441030).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (Mar. 2020a). "Blu: What GUIs Are Made Of." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. IUI '20. <https://doi.org/10.1145/3379336.3381497>. New York, NY, USA: Association for Computing Machinery, pp. 81–82. ISBN: 978-1-4503-7513-9. DOI: [10.1145/3379336.3381497](https://doi.org/10.1145/3379336.3381497).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (Mar. 2020b). "Syn: Synthetic Dataset for Training UI Element Detector From Lo-Fi Sketches." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. IUI '20. <https://doi.org/10.1145/3379336.3381498>. Cagliari, Italy: Association for Computing Machinery, pp. 79–80. ISBN: 978-1-4503-7513-9. DOI: [10.1145/3379336.3381498](https://doi.org/10.1145/3379336.3381498).
- Pandian, Vinoth Pandian Sermuga, Sarah Suleri, and Matthias Jarke (Mar. 2021). "SynZ: Enhanced Synthetic Dataset for Training UI Element Detectors." In: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion*. IUI '21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3397482.3450725](https://doi.org/10.1145/3397482.3450725).
- Pandian, Vinoth Pandian Sermuga and Sarah. Suleri (2020b). "NASA-TLX Web App: An Online Tool to Analyse Subjective Workload." In: *arXiv preprint arXiv:2001.09963*. arXiv: [2001.09963](https://arxiv.org/abs/2001.09963).

- Pillai, Ajit G, Naseem Ahmadpour, Soojeong Yoo, A Baki Kocaballi, Sonja Pedell, Vinoth Pandian Sermuga Pandian, and Sarah Suleri (2020). “Communicate, Critique and Co-Create (CCC) Future Technologies through Design Fictions in VR Environment.” In: *Companion Publication of the 2020 ACM Designing Interactive Systems Conference*, pp. 413–416.
- Rahman, Soliha, Vinoth Pandian Sermuga Pandian, and Matthias Jarke (Mar. 2021). “RUIITE: Refining UI Layout Aesthetics Using Transformer Encoder.” In: *Proceedings of the 26th International Conference on Intelligent User Interfaces Companion*. IUI '21. New York, NY, USA: Association for Computing Machinery. DOI: [10.1145/3397482.3450716](https://doi.org/10.1145/3397482.3450716).

Version: 1.0 as of April 28, 2022

For updates, see <https://vinoth.info/phd>