Journal of Universal Computer Science, vol. 26, no. 9 (2020), 1095-1127
submitted: 5/6/2020, accepted: 27/9/2020, appeared: 28/9/2020 CC BY-ND 4.0

Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques

Daniel de Souza Baulé
(Federal University of Santa Catarina, Florianópolis, Brazil
daniel.baule@grad.ufsc.br)

Christiane Gresse von Wangenheim
(Federal University of Santa Catarina, Florianópolis, Brazil
https://orcid.org/0000-0002-6566-1606, c.wangenheim@ufsc.br)

Aldo von Wangenheim
(Federal University of Santa Catarina, Florianópolis, Brazil
https://orcid.org/0000-0003-4532-1417, aldo.vw@ufsc.br)

Jean C. R. Hauck
(Federal University of Santa Catarina, Florianópolis, Brazil
https://orcid.org/0000-0001-6550-9092, jean.hauck@ufsc.br)

Abstract: The manual transformation of a user interface design into code is a costly and time-consuming process. A solution can be the automation of the generation of code based on sketches or GUI design images. Recently, Machine Learning approaches have shown promising results in detecting GUI elements for such automation. Thus, to provide an overview of existing approaches, we performed a systematic mapping study. As a result, we identified and compared 20 approaches, that demonstrate good performance results being considered useful. These results can be used by researchers and practitioners in order to improve the efficiency of the GUI design process as well as continue to evolve and improve approaches for its support.

Keywords: user interface design, machine learning, systematic mapping
Categories: H.5.2, I.2.6, I.2.10

1 Introduction

Modern software systems rely on attractive graphical user interfaces (GUIs) that facilitate the effective and efficient completion of tasks and engage users, especially in the competitive mobile application market [Moran et al. 2018]. Consequently, GUI design has gained increasing importance in the software process. Typically, the development of GUI involves iterative prototyping, starting with the creative process of making sketches, that are simple hand-drawn representations. They are an effective visual medium to transmit and discuss ideas and compare different alternatives in a simple, quick, and inexpensive way [Huang et al. 2019]. From sketches, wireframes that define the visual hierarchy are created, representing the interface layout and structure without visual design details such as colors, images, etc. [Robinson 2018]. Once the wireframe is created and revised, it is enhanced by the visual design, until it becomes a high-fidelity prototype [Robinson 2018]. When the design is finalized, it is implemented, resulting in the product (Figure 1).

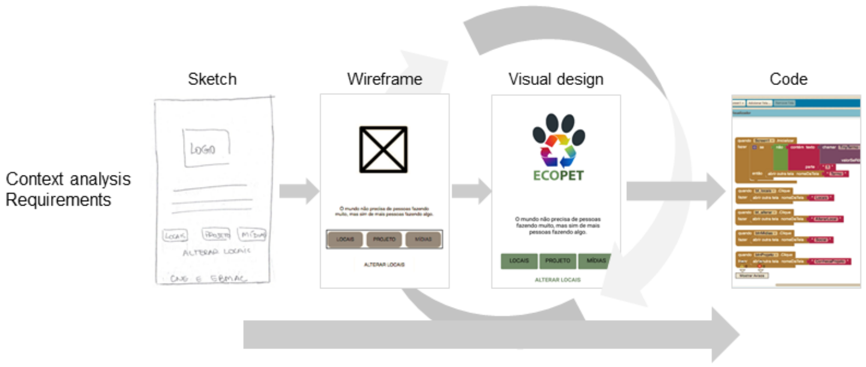


Figure 1: Artefacts in the GUI design process (based on [Garrett 2010] [Hartson and Pyla 2019] [Memmel and Reiterer 2008] [Silva da Silva et al. 2011])

As an iterative prototyping process, these steps can be repeated several times during the design of an interface, requiring rework whenever changes are made. In this context, specifically, the process of the generation of code from GUI prototypes has a high potential for automation, as it is an uninspired, time-consuming, and error-prone task [Moran et al. 2018] [Silva da Silva et al. 2011] [Suleri et al. 2019].

While designers typically used graphic editors including Photoshop or Illustrator to design GUIs, a large variety of design tools (such as Sketch, Figma, Marvel or Adobe XD) have evolved to become all-in-one tools from designing, prototyping to testing, yet, still requiring the coding of the GUI design. On the other hand, modern IDEs such as Eclipse, Visual Studio, or Android Studio, have powerful interactive drag-and-drop based builders for GUI code. But, even with the currently available tools, the transition from sketches or wireframes to code still consists of manually recreating user interfaces [Beltramelli 2019].

Therefore, solutions are being created for the automatic generation of code for the GUIs of websites and mobile applications [Robinson 2018] [Beltramelli 2017]. These tools automatically convert hand-drawn sketches or wireframes into front-end code or code representations. This automation facilitates the GUI development process saving effort and time as well as helping to prevent accidental mistakes [Ozkaya 2019].

More recently, Machine Learning approaches are being applied for such an automatic generation of GUI code. In this context, first solutions for the automatic detection of user interface elements in sketches and or images of GUI prototypes have emerged as an initial step for automating the creation of GUI skeletons and, consequently, front-end code. Yet, although object detection is a vast field of Computer Vision (CV) research, so far research on the detection of GUI elements is still scarce. And, although first approaches have emerged, there still does not exist an overview of existing approaches. Therefore, in this article, we present a systematic mapping concerning the research question on how the generation of GUI design code can be automated based on GUI images using Machine Learning. This article summarizes the main findings from our review that might be of interest to practitioners and researchers.

2 Background

2.1 GUI Design

Typically, the development of GUI involves iterative prototyping, and the nature of GUI design prototypes can vary significantly with respect to fidelity, depending on the design situation, especially the stage of development [Hartson and Pyla, 2019]. And, although terminology with respect to GUI design prototypes differs and is sometimes used interchangeably, a common understanding associates hand-drawn sketches to low-fidelity prototypes and wireframes to medium-fidelity prototypes, representing the skeleton of the GUI by indicating visual elements and layout [Robinson 2018]. Enhancing the wireframe by visual design it becomes a high-fidelity prototype, often also called mockup [Moran et al. 2018], which is generally represented by an image of the GUI and/or a screenshot of the prototype [Robinson 2018].

The user experience results from decisions made during the GUI design process on different planes of details [Garrett 2010]. This includes the representation of hierarchy including GUI elements and structure as part of low fidelity prototypes, such as sketches and wireframes. While on a higher surface plane, visual design details such as color, imagery, and typography are added [Schlatter and Levinson 2013].

The specific GUI elements vary depending on the type of GUI (web or mobile), but typically include elements such as Text, Button, and TextBox among others (Table 1).

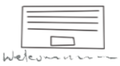
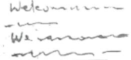
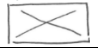
GUI element type	Sketch
Button	
Checkbox	
Dropdown list	
Text field	
Switch	
Sliders	
Icons	
Notifications	
Text	
Image	

Table 1: Examples of types of GUI elements (based on [Garrett 2010]) and their representation in sketches

A GUI design image depicts the desired GUI elements and their spatial layout. To implement a GUI design image this depiction of the interface is typically translated into a GUI skeleton, which defines what and how the components of a GUI builder should be composed in the GUI implementation for reproducing the GUI elements and their spatial layout in the GUI design image. This GUI skeleton then enables the subsequent GUI implementation depending on the specific platform and/or programming environment during front-end development.

Websites are composed structurally with HTML element tags and styled with Cascading Style Sheets (CSS). The structure of a website consists of the types of elements e.g. division, table, paragraph, button, and how they are positioned, while the style defines the colors, fonts, and borders. HTML is constructed as a tree of objects (Document Object Model (DOM)), with branches of this tree representing containers, such as <div>, <footer>, <header>, and leaves of the tree are elements which contain content, e.g., <img>, <p>, <button>.

The way mobile GUIs are defined varies depending on the technology used, which can be layered on top of the basic Domain-Specific Language (DSL) definition infrastructure. However, in general, regardless of the overlapping technologies, all elements of the user interface in an Android application, for instance, will ultimately be implemented in the form of tree-like structures, with each node representing View objects, which draws something on the screen and allows user interaction, and ViewGroup, which groups those elements. GUI elements are defined in a markup-like format (XML) using several View subclasses, or "widgets", which can be used for input control (such as buttons: <Button> and text fields: <TextView>) or even ViewGroup, or "layouts", such as a linear layout <LinearLayout>, for example. Other aspects such as color, font size, and background can be defined as styles and themes, also in a markup-like (XML) format. Styles can then be applied to specific attributes of View objects, such as TextView, using attributes like "style".

2.2 Machine Learning Approaches for Object Detection

Object detection a fundamental and challenging problem in computer vision has received great attention in recent years is due to its wide range of real-world applications, including the detection of elements in graphical user interfaces [Zou et al. 2019]. Especially Machine Learning approaches have achieved remarkable advances using pattern recognition techniques as well as deep learning [Jiao et al. 2019].

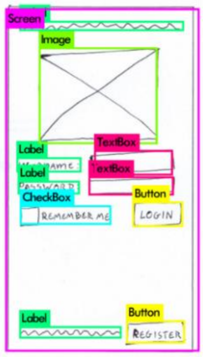


Figure 2: Example of GUI element detection in sketches

Object detection deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. In the context, of GUI design, it aims at the detection of GUI elements such as buttons, text, and images (Figure 2), referring to a multi-class detection, where each detected object in the GUI is classified into one of the non-overlapping GUI element classes.

2.2.1 Object detection process

Following [Hechun and Xiaohong 2019], a basic process for object detection can roughly be divided into the following phases: region proposal, feature representation, and region classification. The first possible object locations in the image are suggested, indicating also some possible candidate area of the containing object. Then features are extracted by selecting the appropriate features as characteristics vector, and classifying this feature vector, and, thus, determining its type. After executing post-processing operations, such as the maxima inhibition, border position return, the final object frame is returned. Some more recent deep learning approaches condense some of these steps.

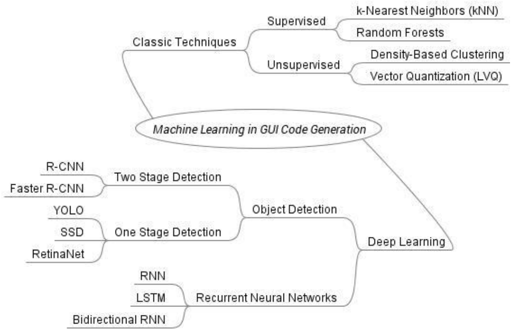


Figure 3: Overview of Machine Learning approaches applied for GUI element detection

2.2.2 Machine Learning approaches for object detection

Different types of learning can be applied [Russell and Norvig 2010] including supervised and unsupervised approaches. Supervised learning focuses on inferring a function that maps an input to an output based on labeled training data. Unsupervised learning, on the other hand, aims at finding previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.

Adopted Machine Learning approaches include k-Nearest Neighbors (kNN) [Zhang 2016], a is a non-parametric method used for classification and regression in pattern recognition. Using supervised learning methods, the kNN algorithm stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). The Learning Vector Quantization algorithm (LVQ) [Kohonen 1988] for pattern classification combines competitive learning with supervision enabling to choose how many training instances to hang onto and learns exactly what those instances should look like. Random Forests [Ho 1995] is a supervised learning method for classification, regression, and other tasks that builds an ensemble of decision trees.

Other examples include Density-Based Clustering, which refers to unsupervised learning methods that identify distinctive groups in the data, assuming that a cluster is a contiguous region of high point density, separated from other groups by contiguous regions of low point density.

2.2.3 Deep Learning approaches for object detection

Recently, Deep Learning approaches have also been used (Figure 3). Deep Learning originated from the study of Artificial Neural Networks (ANNs), which are computation models inspired by biological neural networks representing a group of multiple perceptrons/neurons at each layer processing inputs. A Deep Learning Neural Network (DNN) is an ANN with two additional features: a very deep structure with many layers between the input and output layer and convolutional layers. In a convolutional layer, input connections of each artificial neuron can be spatially organized, e.g., reflecting a field of pixels in an image, and the input signal is always subject to a convolution operation, i.e., a matrix operation that is applied to the input, instead of the weighted sum of inputs of a traditional ANN. Convolution operations can be understood as filters on the input signal. Examples of classic convolution operations are Gaussian filters, Perona-Malik diffusion filters, and gradient operators. A DNN that employs convolutional layers is also called a Convolutional Neural Network (CNN). CNNs do not employ ready-made filters, but rather learn customized, new filters during the training process. CNNs were first proposed by [LeCun et al. 1998], but have been applied more widely only after [Krizhevsky et al. 2012] demonstrated their power and the possibilities offered by training them on general-purpose graphic processing units (GP-GPU).

CNN are used heavily for object detection [Zhao et al. 2019]. Currently, CNN-based object detection approaches can be primarily divided into two types: structured, two-stage detectors, such as Region-based CNN (R-CNN) and its variants (such as Faster R-CNN) and one-stage, purely neural detectors, such as SSD, RetinaNet and YOLO and their variants [Wu et al. 2020]. Two-stage detectors first generate regions of interest, sometimes employing classical methods, and then extract features from each proposal, followed by region classifiers that predict the specific category of the proposed region. These models reach the highest accuracy rates but are much slower. One-stage detectors, on the other hand, directly make a categorical prediction of objects on each location of the feature maps without the region classification step. They treat object detection as a simple regression problem by taking an input image and learning the class probabilities together with the bounding box coordinates. Such models reach lower accuracy rates, but are much faster and can be used in real-time applications [Huang et al. 2017].

Classic Recurrent Neural Networks (RNN) are a variant of ANNs that redirect part of the output of some layers back into the input of earlier layers, providing a means of learning contextual information, such as temporal and sequential data and also features in their spatial contexts [Williams and Zipser 1989]. They perform the same task for every element of a sequence, with the output being dependent on the previous computations. Modern, Deep Learning-based RNNs [Sherstinsky 2020] are a variant of CNNs used mainly in signal analysis and Natural Language Processing (NLP). RNNs have shown to be hugely successful in NLP, especially with a variant called long-short-term memory (LSTM), which is able to look back longer than RNNs. LSTM is a special kind of structured RNN, with different recurrent modules capable of learning long-term dependencies. Another alternative are Bi-directional RNNs, stacking two LSTM RNNs, which allows the output layer to integrate information from both, past and future states.

Nowadays, there exist a large number of Deep Learning frameworks, which offer basic and advanced Deep Learning features and can be easily extended and used as building blocks for the development of new CNN models and applications. The most widely used are TensorFlow, PyTorch, Keras, and Fastai, which employ the Python programming language, and Darknet and OpenCV-DNN, which employ the C++ programming language.

A backbone network is a CNN model employed in a wider CNN-based architecture, where it acts as the basic feature extractor for object detection, semantic segmentation, or automated image captioning tasks. The backbone network is generally an image classification CNN without its last classification layers, which takes images as input and outputs feature vectors of that image [Jiao et al. 2019]. Recent research on CNNs has focused on wider, structured modular architectures for more complex tasks and employed well-known and proved CNNs as their backbone networks. Depending on requirements on accuracy vs. efficiency, either deeper backbone networks, like ResNet or lightweight backbone networks like MobileNet or Xception can be chosen.

2.2.4 Datasets

For object detection there exist several well-known datasets and benchmarks, including the datasets of PASCAL VOC Challenges, ImageNet Large Scale Visual Recognition Challenge, or the MS-COCO Detection Challenge [Zou et al. 2019]. In addition to general image datasets, specific datasets have been created for applications in certain areas, including GUI image datasets such as RICO [Deka et al. 2017] that contains GUI design data from more than 9.3k Android apps spanning 27 categories. For each app, Rico exposes Google Play Store metadata, a set of user interaction traces, and a list of more than 66k unique GUI screens discovered. Each GUI comprises a screenshot, an augmented Android view hierarchy exposing for all GUI elements visual, textual, structural, and interactive design properties, as well as a set of explored user interactions, a set of animations capturing transition effects in response to user interaction, and a learned vector representation of the GUI’s layout. Another example is Syn [Pandian et al. 2020a], a dataset containing 125,000 synthetically generate GUI sketches, which has been generated using the UISketch dataset containing 5,917 sketches including 19 GUI elements drawn by 350 participants.

The evaluation of object detection models is not trivial, as it requires to measure two distinct factors: (i) determining whether an object exists in the image (classification) and (ii) determining the location of the object (localization). Furthermore, in the context of multi-class object detection, there are many classes, which may not be uniformly distributed. Consequently, an accuracy-based metric may introduce biases.

2.2.5 Performance evaluation in object detection

The performance of object detection models is typically based on the Intersection Over Union (IOU), a measure based on the Jaccard Index that evaluates the overlap between two bounding boxes, the ground truth bounding box and the predicted bounding box [Zou et al. 2019]. Generally, a 0.5 IOU ratio for each prediction at the training stage is aimed for, which means that if the model predicts an object with a bounding box that overlaps with the ground truth box by at least 50%, it is considered as a true prediction. By applying the IOU, results can be classified into:

- **True Positive (TP):** correct detection with IOU ≥ threshold
- **False Positive (FP):** erroneous detection with IOU < threshold
- **False Negative (FN):** ground truth not detected

Depending on the kind of model, popular metrics for the evaluation of object detection models are summarized in Table 2 [Sokolova and Lapalme 2009][Lie et al. 2020].

However, in recent years, the most frequently used metric for object detection is Average Precision (AP) [Liu et al. 2020]. AP is defined as the average detection precision under different recalls and is usually evaluated in an object class-specific manner. To compare performance overall object classes, the mean AP (mAP) averaged over all object classes is commonly used as the final metric of performance.

Category	**Metric**	**Definition**
Classification metrics	Accuracy	**Number of correct predictions divided by the total number of predictions,** multiplied by 100
	Precision	**Precision= TP/ (TP+ FP)**
	Recall	**Recall= TP/ (TP+ FN)**
	F1 score	**F1 score= 2*Precision*Recall/(Precision+Recall)**
Regression metrics	Mean squared error (MSE)	Average squared error between the predicted and actual values
	Mean absolute error (MAE)	Average absolute distance between the predicted and target values

Table 2: Examples of metrics for the evaluation of object detection models

Among metrics to measure the similarity of images, the simplest and most widely used is the mean squared error (MSE), averaging the squared intensity differences of distorted and reference image pixels, along with the related quantity of peak signal-to-noise ratio (PSNR). Yet, it may not be very well matched to perceived visual quality [Wang et al. 2004]. Another metric is the Structural Similarity Index (SSIM) that measures the perceptual difference between two similar images. It attempts to separate the task of similarity measurement into luminance, contrast, and structure [Wang et al. 2004]. Also, metrics originating from the evaluation of machine translation/summarization are used. These are typically also used for the evaluation of image captioning models that mainly measure the word overlap between generated and reference captions, including Bilingual Evaluation Understudy (BLEU), Recall Oriented Understudy for Gisting Evaluation (ROUGE), Metric for Evaluation of Translation with Explicit Ordering (METEOR), among others [Aafaq et al. 2019][Ciu et al. 2018]. BLEU is a precision-based metric based on the precise matching of n-grams in the generated and ground truth artifacts. METEOR creates an alignment between the two artifacts by comparing elements. ROUGE, uses different n-grams based versions to BLEU and computes recall. Another similarity measure used is the Levenshtein distance measure [Yujian and Bo 2007], computing the number of operations needed to transform one string into another string, usually limiting the possible operations to insertion, deletion, and substitution.

3 Research Method

In order to provide an overview of the state of the art on the automatic generation of code from sketches or graphical user interface (GUI) images, we performed a systematic mapping of the literature following the procedure defined by [Petersen et al. 2015].

3.1 Definition of the Review Protocol

The objective of this systematic mapping is to answer the research question: What approaches exist for the automatic generation of code from GUI images using Machine Learning (ML)?

This research is refined in the following analysis questions in order to answer the research question and to characterize and compare the existing approaches:
AQ1. What approaches for generating code from GUI images exist?
AQ2. For which kind of platform and input/output data are the approaches?
AQ3. Which data sets are used and what are their characteristics?
AQ4. What are the characteristics of the ML models?
AQ5. How was the quality of the result evaluated and which results were obtained?

Inclusion and exclusion criteria. We consider only English-language publications that present a model for generating code or GUI design representations based on sketches or images of GUIs created during the GUI design process. We only consider approaches adopting Machine Learning, not including models based on other techniques. We only consider studies related to images of user interfaces of software systems (web and mobile) that have been published since 2010. We consider only articles that present substantial information allowing the extraction of relevant information regarding the analysis questions. Therefore, abstract-only or one-page articles are excluded.

Sources. We searched the main digital databases and libraries in the field of computing, including ACM Digital Library, IEEE Xplore Digital Library, arXiv.org e-print archive, and Scopus with access via the Capes Portal[1]. Based on the research question, several informal searches were performed to calibrate the search string, identifying relevant search terms and their synonyms (Table 3). Synonyms were used to minimize the risk of omitting relevant works.

Keyword	Synonym(s)
sketch	sketch, mockup, screenshot
wireframe	wireframe
user interface	user interface, ui, gui
software system	app, mobile, android, ios, website
machine learning	deep learning, neural network, cnn, computer vision

Table 3: Search terms and respective synonyms

[1] A web portal for access to scientific knowledge worldwide, managed by the Brazilian Ministry on Education for authorized institutions, including universities, government agencies and private companies (www.periodicos.capes.gov.br).

As a result of the calibration, a specific search string was defined in accordance with the specific syntax of each of the data sources (Table 4).

Repository	Search string
ACM Digital Library	(sketch* OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface*" OR GUI) AND (app* OR website* OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision")
IEEE Xplore Digital Library	(sketch* OR wireframe* OR screenshot* OR mockup*) AND (ui OR "user interface" OR "user interfaces" OR GUI) AND (app* OR website OR websites OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network" OR "neural networks" OR cnn OR "computer vision")
Scopus	TITLE-ABS-KEY ((sketch* OR wireframe* OR screenshot* OR mockup*) AND [ui OR "user interface*" OR GUI) AND (app* OR website* OR ios OR mobile OR android) AND ("machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision")) AND PUBYEAR > 2010
arXiv.org e-print archive	order: -announced_date_first; size: 50; date_range: from 2010-01-01 ; include_cross_list: True; terms: AND all=sketch* OR wireframe* OR screenshot* OR mockup*; AND all=ui OR "user interface*" OR GUI; AND all=app* OR website* OR ios OR mobile OR android; AND all="machine learning" OR "deep learning" OR "neural network*" OR cnn OR "computer vision"

Table 4: Search strings used in the different repositories

3.2 Execution of the Search

The search was executed in April 2020 by the authors. The initial search resulted in a total of 2,576 search results (Table 5).

Repository	Quantity of search results	Quantity of potentially relevant articles (based on title and abstract)	Quantity of relevant articles (based on full-text analysis)
ACM Digital Library	2,516	34	8
IEEE Xplore Digital Library	18	17	3
Scopus	39	35	8
arXiv.org e-print archive	3	3	3
Total	2,576	89	16 (without duplicates)

Table 5: Overview of the search results and selection process

During the first stage, the search results were quickly analyzed based on their title and abstract. Irrelevant and duplicate papers returned my multiple searches were removed. This stage left us with 89 potentially relevant articles. During the second stage of selection, we analyzed the full text of the articles applying the inclusion and exclusion criteria to identify relevant articles.

Research using different inputs than sketches or images of GUIs, such as natural language requirements [Kolthoff 2019][Sethi et al. 2019] were not considered. We also included only articles dealing with sketches of user interfaces, excluding any

sketch of another kind of object [Huang and Canny 2019]. Articles presenting only data sets have been excluded [Pandian et al. 2020a][Deka et al. 2017]. We also excluded articles that focus exclusively on the detection of specific GUI elements, such as icons [Xiao et al. 2019]. We excluded research directed rather on the retrieval of GUIs [Chen et al. 2019b] [Huang et al. 2019], generation of GUI design patterns [Nguyen et al. 2018], or semantic annotations [Liu et al. 2018a] rather than code. On the other hand, we have included the article [Ge 2019] focusing primarily on the search for visually similar apps based on sketches, as it, as part of the retrieval process, also generates intermediate GUI prototypes representing visual elements and layout. In accordance with our focus, we excluded articles that do not use any kind of Machine Learning technique [Nguyen and Csallner 2015]. And, although there exist already commercial tools such as Airbnb's sketching interfaces tool (https://airbnb.design/sketching-interfaces), Microsoft AI.Lab's Sketch2Code (https://www.microsoft.com/en-us/ai/ai-lab-sketch2code), TeleportHQ (https://teleporthq.io) and Zecoda (https://zecoda.com), which support the conversion of sketches into code using Machine Learning, we did not include them as no detailed information on these tool has been encountered.
Based on the primary studies, we also identified and added relevant secondary literature, including [Beltramelli 2019][Halbe and Joshi 2015][Pandian and Suleri 2020] [Kumar 2018][Wallner 2018]. As a result, a total of 21 relevant articles has been identified representing 20 approaches (Table 6).

According to the analysis questions relevant data has been extracted from the articles. Data extraction was done independently by two of the authors and then revised by all authors until consensus was obtained. Varying terminology referring to the same concept has been unified and aggregated.

4 Results

The results of the data analysis are presented for each of the analysis questions.

4.1 What Approaches for Generating Code from GUI Images Exist?

Reference		**Description**
[Aşıroğlu et al. 2019]	B.Aşıroğlu et al. Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques. Proc. of the Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science, Istanbul, Turkey, 2019.	Approach for the automation of the code generation process from hand-drawn mock-ups using CV techniques and deep learning methods.
[Bajammal et al. 2018]	M. Bajammal et al. Generating reusable web components from mockups. Proc. of the 33rd ACM/IEEE Int.Conference on Automated Software Engineering, Montpellier, France, 2018.	A tool to automate the generation of reusable web components from a mock-up employing visual analysis and unsupervised learning of visual cues to create reusable web components.
[Beltramelli 2019]	T. Beltramelli. Hack your design sprint: wireframes to prototype in under 5 minutes. Medium, 2019. https://uizard.io/	Uses CV and ML to automatically transform wireframe images to high-fidelity mock-ups that can be exported to front-end code such as HTML/CSS.
[Beltramelli 2018]	T. Beltramelli. pix2code: Generating Code from a Graphical User Interface Screenshot. Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Paris, France, 2018.	An approach based on CNN and RNN allowing the generation of computer tokens from a single GUI screenshot as input.
[Chen et al. 2018]	C. Chen et al. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. Proc. of the Int. Conference on Software Engineering, Gothenburg, Sweden, 2018.	A deep learning architecture combining CNN and RNN models that distills the crowd-scale knowledge of GUI designs and implementations from existing apps to develop a generative tool to automatically generate a GUI skeleton given a GUI

		image.
[Chen et al. 2019a]	S. Chen et al. Automated cross-platform GUI code generation for mobile apps. In Proc. of the 1st Int. Workshop on Artificial Intelligence for Mobile, Hangzhou, China, 2019.	An automated cross-platform GUI code generation framework using image processing and deep learning classification techniques to transfer the GUI code implementation between two mobile platforms.
[Ge 2019]	X. Ge. Android GUI search using hand-drawn sketches. Proc. of the 41st Int. Conference on Software Engineering, Piscataway, USA, 2019.	An approach that searches for visually similar apps using sketches using Deep Learning to translate sketches into GUI structures. Then, similar GUIs are identified by computing a similarity score between structural GUI data with the ones in the app repositories.
[Halbe and Joshi 2015]	A. Halbe and. R.Joshi. Novel Approach to HTML Page Creation Using Neural Network. Procedia Computer Science, 45, 2015.	An approach to create an HTML page automatically from a hand-drawn paper sketch. The system segments the various HTML controls, which are then identified using ML. Identified HTML controls are stored in an XML database that contains the name and position of the component on the GUI design. Then, this XML file is parsed to generate an HTML page.
[Han et al. 2018]	Y. Han et al. CSSSketch2Code: An Automatic Method to Generate Web Pages with CSS Style. Proc. of the 2nd Int. Conference on Advances in Artificial Intelligence, Barcelona, Spain, 2018	A method based on object detection and attention mechanism to automatically generate a web page with CSS style information.
[Huang et al. 2018]	R. Huang et al. Automatically Generating Web Page from A Mockup. Proc. of the 28th Int. Conference on Software Engineering and Knowledge Engineering, Redwood City, CA, USA, 2016.	A method to automate the transformation of a mockup into a web page by extracting the elements based on the color features of the edges. A bottom-up tag generating method based on Random Forest is proposed to select the tags for elements. The web page is generated by the definition of HTML/CSS code.
[Jain et al. 2019]	V. Jain et al. Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network. arXiv:1910.08930 [cs.CV], 2019.	An approach that employs a DNN to detect GUI elements in sketches. The output is a platform-independent UI representation object used by a GUI parser which creates code for different platforms.
[Kim et al. 2018]	B. Kim et al. Deep-Learning based Web UI Automatic Programming. Proc. of Int. Conference on Research in Adaptive and Convergent Systems, Honolulu, USA, 2018.	Recognizing web layout based on hand-drawn sketches using computer vision algorithms and web widgets using Faster R-NN, the approach generates HTML code automatically.
[Kumar 2018]	A. Kumar. Automated front-end development using deep learning. Medium “Insight”, 2018.	An approach based [Beltramelli 2018] and [Wallner 2018] to generate HTML code based on hand-drawn website sketches.
[Liu et al. 2018b]	Y. Liu et al. Improving pix2code based Bi-directional LSTM. Proc. of the IEEE Int. Conference on Automation, Electronics and Electrical Engineering, Shenyang, China, 2018.	A framework based on deep learning using CNN and LSTM to transform a GUI screenshot into code. The model is optimized by a Bidirectional LSTM.
[Moran et al. 2018]	K. Moran et al. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. IEEE Transactions on Software Engineering, 46[2], 2018.	An approach that first detects logical components of a GUI from a mock-up artifact using CV or mock-up metadata. Then, by software repository mining, automated dynamic analysis, CNNs are used to classify GUI-components into domain-specific types. A data-driven, K-nearest-neighbors algorithm generates a suitable hierarchical GUI structure from which a prototype application can be automatically assembled.
[Pandian et al. 2020b]	V. P. S. Pandian et al. Blu: What GUIs are made of. Proc. of the 25th Int. Conference on Intelligent User Interfaces, 2020. https://blu.blackbox-toolkit.com/	An approach that uses deep learning and gestalt laws-based algorithms to convert GUI screens to editable blueprints by identifying the constituent GUI element categories, their location, dimension, text content, and layout hierarchy.
[Robinson 2019]	A. Robinson. Sketch2code: Generating a website from a paper mockup, Dissertation, University of Bristol, UK, 2019.	Using an ANN to translate a wireframe into a normalized image.
[Suleri et al. 2019] [Pandian and	S. Suleri et al. Eve: A Sketch-based Software Prototyping Workbench. Proc. of the Conference on Human Factors in Computing Systems Extended Abstracts, Glasgow, UK, 2019.	A prototyping workbench that automatically generates code based on sketches. It generates MeFi and HiFi prototypes using GUI element detection (MetaMorph) created with a DNN model that

Suleri 2020]	V. P. S. Pandian and S. Suleri. BlackBox Toolkit: Intelligent Assistance to UI Design. Proc. of the Workshop on Artificial Intelligence for HCI: A Modern Approach, Honolulu, USA, 2020. https://metamorph.designwitheve.com/	detects constituent GUI element categories and their position and code generation. With this information, the respective GUI elements are created as a medium-fidelity prototype. Lastly, the Code Generator transforms the MeFi to HiFi by generating executable code.
[Wallner 2018]	E. Wallner. Turning Design Mockups into Code with Deep Learning, Floydhub, 2018.	Approach using ML to code a basic HTML and CSS website based on a picture of a design mockup.
[Yun et al. 2018]	Y. Yun. Detection of GUI elements on sketch images using object detector based on deep neural networks. Proc. of the 6th Int. Conference on Green and Human Information Technology, Chiang Mai, Thailand, 2018.	Approach adopting object detection based on DNN that finds GUI elements by the integration of localization and classification.

Table 6: Relevant research

Most of the relevant articles have been published during the last 3 years (Figure 4), indicating the recent increased interest in this topic.

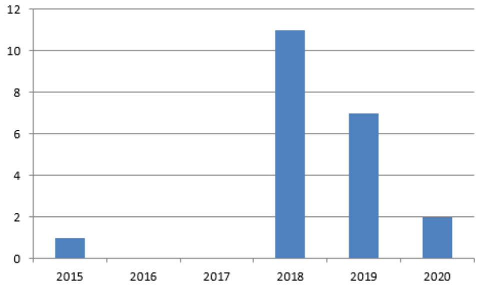


Figure 4: Quantity of articles per year

4.2 For Which Kind of Platform and Input/ Output Data are the Approaches?

The approaches target web and mobile platforms (Figure 5). Most approaches aim at the design of web GUIs, followed by Android GUIs. A smaller number of approaches targets iOS applications. Several researches present a multi-platform approach, with two approaches covering both prominent mobile application platforms and four approaches including web and mobile platforms.

Most approaches are based on hand-drawn sketches of GUI as input rather than high-fidelity GUI design images or screenshots (Figure 6).

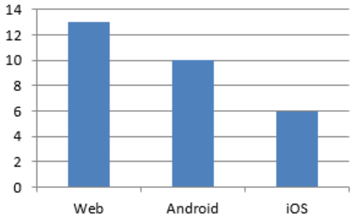


Figure 5: Distribution of approaches per platform

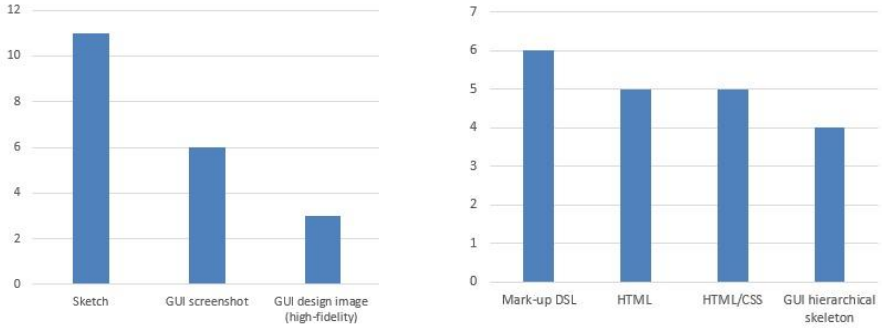


Figure 6: Distribution of approaches per input and output type

For many approaches, no detailed information on which types of GUI elements are detected has been encountered. Yet, to simplify the detection some approaches limit the elements to be detected to only a small set. For example, [Aşıroğlu et al. 2019] only considers four different types of elements such as TextBox, Dropdown, Button, and CheckBox. Similarly, [Robinson 2019] also considers only four types of GUI elements and [Ge, 2019] only seven. [Jain et al. 2019] concentrates on the ten most frequent GUI elements, whereas [Moran et al. 2018] extends this to the 15 most used GUI elements. [Suleri et al. 2019] consider 19 of Material Design's GUI elements. The largest number (25) of considered GUI elements is reported by [Pandian et al. 2020b].

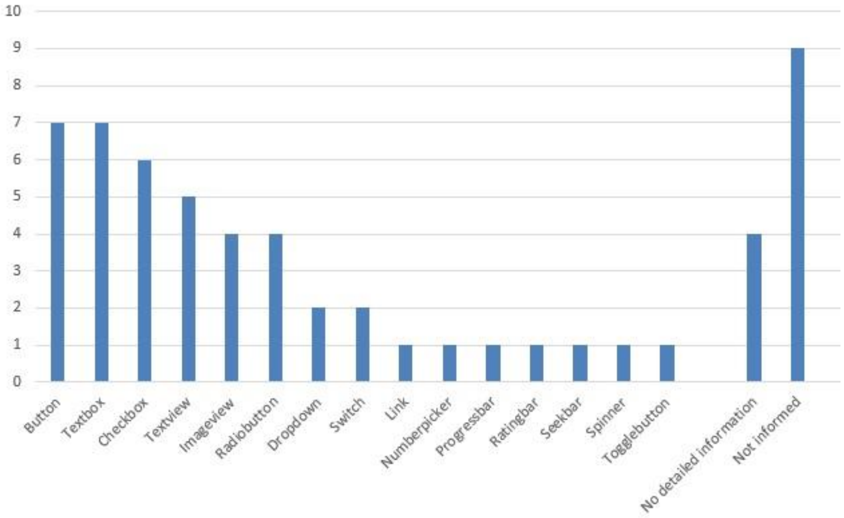


Figure 7: Frequencies of consideration of GUI element types for detection

The analysis of the frequencies per GUI element also indicates this focus typically on the most used GUI elements, such as Textbox, Button, etc., and less on elements used with less frequency such as Spinner or NumberPicker. We also observed that most approaches consider texts in general, without a more detailed detection per type of text. An exception is the model presented by [Jain et al.2019] that separately detects Paragraph, Label, and Heading as well as [Robinson 2019] separating Title and Paragraph.

And, although, four models aim at detecting images, no further details on imagery in general and icons specifically are presented, although this seems to be an important issue concerning GUI design also approached by other research, such as [Liu et al. 2018b] [Xiao et al. 2019].

Reference	**Platform**	**Type of input**	**Type of output**	**Detected GUI elements**
[Aşıroğlu et al. 2019]	Web	Sketch	HTML	Four different types of components such as textBox, dropdown, button, and checkBox
[Bajammal et al. 2018]	Web	GUI design image (hi-fi)	HTML	NI
[Beltramelli 2019]	Android, iOS and Web	Sketch	HTML/CSS High-fidelity prototype (HTML/CSS)	NI
[Beltramelli 2018]	Android, iOS, and Web	GUI screenshot	Markup-like DSL Code in markup-like DSL	NI
[Chen et al. 2018]	Android	GUI design image (hi-fi)	Markup-like DSL GUI framework language whose vocabulary consists of the component names, such as Android's RelativeLayout, TextView, etc.	NI
[Chen et al. 2019a]	Android, iOS	GUI screenshot	Markup-like DSL Android or iOS code	NI
[Ge 2019]	Android	Sketch	GUI hierarchical skeleton: tree-like structure with each node representing a widget or a layout (JSON/APK)	TextView, EditText, ImageView, Button, RadioButton, Switch, and CheckBox
[Halbe and Joshi 2015]	Web	Sketch	HTML	NI
[Han et al. 2018]	Web	Sketch	HTML	NI
[Huang et al. 2018]	Web	GUI screenshot	HTML/CSS	DIV, P, LI, UL, H, FORM, IMG, INPUT
[Jain et al. 2019]	Android, iOS, and Web	Sketch	HTML	10 most used classes of GUI elements: Link, Image, Paragraph, CheckBox, TextBox, SelectBox, Label, Heading, RadioButton, Button
[Kim et al. 2018]	Web	Sketch	HTML/CSS	Button, RadioButton, CheckBox, Textbox, Text, etc.
[Kumar 2018]	Web	Sketch	HTML/CSS	16 GUI elements, such as buttons, text boxes, and divs
[Liu et al. 2018b]	Android, iOS, and Web	GUI screenshot	Markup-like DSL	NI
[Moran et al. 2018]	Mobile	Mock-up artifact hi-fi (with or without meta-data)	GUI hierarchical skeleton	15 most popular elements: TextView, ImageView, Button, ImageButton, EditText, CheckedTextView, CheckBox, RadioButton, ProgressBar, SeekBar, NumberPicker, Switch, ToggleButton, RatingBar e Spinner
[Pandian et al. 2020b]	Android	GUI screenshot	GUI hierarchical skeleton	25 categories of GUI elements
[Robinson 2019]	Web	Sketch	GUI hierarchical skeleton JSON tree-like structure used to represent the structure of a	Title, Image, Button, Input e Paragraph

			wireframe that can directly be translated into HTML	
[Suleri et al. 2019][Pandian and Suleri 2020]	Android	Sketch	Markup-like DSL XML code	19 Google's Material Design based GUI elements, such as buttons, text fields, menus, etc.
[Wallner 2018]	Web	GUI screenshot	HTML/CSS	17 simplified tokens that are translated into HTML and CSS
[Yun et al. 2018]	NI	Sketch	Markup-like DSL XML code	NI

Table 7: Characteristics of platform and input/output

Most of the approaches focus solely on detecting the type of GUI element and position, not aiming at the understanding of handwritten text in order to automatically set the text values of these GUI elements in the generated code. Approaches that extract handwritten content from text GUI elements, such as labels and buttons, typically use Optical Character Recognition (OCR). For example [Moran et al. 2018] and [Pandian et al. 2020b] use the open-source Tesseract OCR library. [Halbe and Joshi 2015] take a different approach, using a separate Learning Vector Queue Neural Network dedicated to the recognition of handwritten uppercase letters. [Han et al. 2018] recognize textual values, using textual data in order to define CSS styles for the output code, such as font and color, yet without providing further details on how the text understanding is performed. [Robinson 2019] performs text detection using Stroke Width Transform, but only to detect the position and size of text, and not its content, using default values instead when generating the GUI code. All other approaches use default/random values for generating the text in labels and buttons, requiring still a manual substitution during the GUI design process.

4.3 Which Datasets Have Been Used for Building the Machine Learning Model?

Only two researches used a pre-existing dataset RICO [Ge 2019] [Pandian et al. 2020b]. On the other hand, several researches systematically created and made their datasets available as part of the research:

- Pix2Code (https://github.com/tonybeltramelli/pix2code/tree/master/datasets)
- Android UI (http://tagreorder.appspot.com/ui2code.html)
- REDRAW (https://zenodo.org/record/2530277)
- UISketch (https://www.kaggle.com/vinothpandian/uisketch)
- Syn-dataset (https://www.kaggle.com/vinothpandian/syn-dataset)

Most of the research works crawled online stores or sites to collect web sites or mobile applications and then used dynamic testing tools to automatically capture the screenshots. In some cases, GUI images have been synthetically generated, for example by the automated exploration of the GUI hierarchies of app screens [Moran et al. 2018] or by generating an editable blueprint vector graphic and a GUI layout tree based on GUI screens and annotations from the RICO dataset [Pandian et al. 2020b]. Others automatically generate sketches based on GUI screenshots, e.g., by using web script/CSS-generated synthetic data [Han et al. 2018] [Kumar, 2018], screenshots [Robinson 2019] or synthetic data generated from hand-drawn sketches [Suleri et al. 2019] [Pandian and Suleri 2020]. In these cases, sketches may vary with respect to their authenticity, representing in some cases rather artificial examples of GUIs and/or GUI elements that may not be representative for real-world interfaces. Yet, this issue and its potential impact on performance and validity are not further discussed in most cases. An exception is Robinson (2019), who reports a reduction of the performance of the deep learning approach when applied to real sketches, emphasizing, thus, the importance of a great variety in sketches to allow the deep learning approach to better generalize to unseen sketching styles.

Other datasets use either sketches that have hand-drawn specifically for this purpose. Some research also used data-augmentation techniques to balance the frequencies of GUI element types in the dataset [Moran et al. 2018]. For supervised learning approaches, the images are commonly annotated manually.

The datasets vary considerably in size (Figure 8). Yet, only two studies use very small datasets, including only 50 [Yun et al. 2018] or 149 sketches [Jain et al. 2019]. The majority uses datasets ranging from 1,500 GUIs [Han et al. 2018] to about 5,250 GUIs [Beltramelli 2018]. Several studies also use large datasets, including the one used by [Chen et al. 2018] representing 1,842,580 unique Android screenshots.

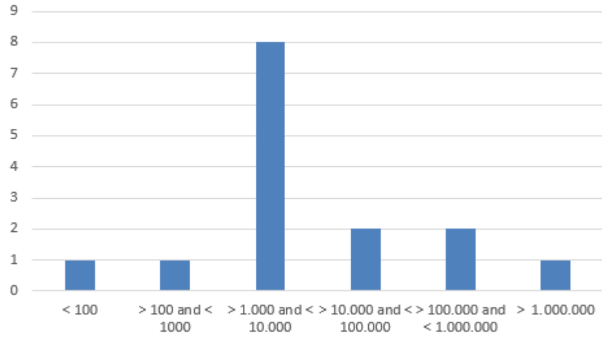


Figure 8: Distribution per category of dataset size

Reference	Name	Description	Size	Link
[Aşıroğlu et al. 2019]	NI	Used some of the images provided from Microsoft AI Lab for their Sketch2Code application.	NI	NI
[Bajammal et al. 2018]	NI	NI	NI	NI
[Beltramelli 2019]	NI	NI	NI	NI
[Beltramelli 2018]	Pix2code dataset	Synthesized set of GUI screenshots and GUI code.	5,250 instances (1,750 Android, 1,750 iOS, 1,750 Web)	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
[Chen et al. 2018]	Android UI dataset	Images from more than 5000 Android Apps crawled from Google Play taking the screenshots as the GUI design image and also collecting the corresponding code automatically using the testing tool Stoat.	185,277 pairs of GUI images and GUI skeletons with 291 unique Android GUI components	http://tagreorder.appspot.com/ui2code.html
[Chen et al. 2019a]		Crawled Android and iOS apps from online stores and used dynamic testing tools for	1,842,580 unique Android screenshots	NI

		taking screenshots during runtime.		
[Ge 2019]	Rico [Deka et al. 2017]	The dataset was created by crowdsourcing and automation to mine design and interaction data from Android apps at runtime providing information on visual, textual, structural, and interactive properties.	Data from more than 9,3k Android apps	NI
[Halbe and Joshi 2015]	NI	NI	NI	NI
[Han et al. 2018]	NI	The dataset involves the original web script, the CSS pixel matrix, and its corresponding DSL label sequences. Photoshop and graphics tablets are used to draw the web scripts and labelImg is used for annotation.	Data on 1,500 web scripts	NI
[Huang et al. 2018]	NI	NI	NI	NI
[Jain et al. 2019]	NI	Sketches of GUI elements drawn by individuals.	149 sketches, containing 2,001 samples of GUI elements	NI
[Kim et al. 2018]	NI	NI	NI	NI
[Kumar 2018]	Pix2code dataset (web only)	Synthesized set of GUI screenshots changing the CSS stylesheets to make them look like hand-drawn sketches and GUI code.	1,750 website screenshots	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
[Liu et al. 2018b]	pix2code (web only)	Used part of the pix2code dataset.	1,750 web GUI	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
[Moran et al. 2018]	REDRAW	Dataset of mobile application GUI data containing screenshots and GUI related metadata.	14,382 GUI screenshots and 191,300 labeled GUI elements	https://zenodo.org/record/2530277
[Pandian et al. 2020b]	Rico [Deka et al. 2017]	Annotated Android GUI screens.	57,775 annotated Android GUI screens containing 25 categories of GUI elements	NI
[Robinson 2019]	NI	The dataset contains sketches and their associated normalized version of the website. Sketches were created based on the normalized and reduced screenshots by extracting elements and automatically replacing them with hand-drawn elements.	Screenshots from 1,750 URLs	NI
[Suleri et al. 2019] [Pandian and Suleri 2020]	UISketch dataset	GUI element sketches were collected from 350 participants using paper and digital questionnaires.	5,906 GUI element sketches of 19 Google material design GUI elements	https://www.kaggle.com/vinothpandian/uisketch
	Syn-dataset [Pandian et al. 2020a]	125,000 synthetically generated sketches by randomly choosing GUI elements from the labeled UISketch dataset and stitching them in random locations with random scaling.	125,000 lo-fi sketches	https://www.kaggle.com/vinothpandian/syn-dataset

[Wallner 2018]	Pix2code dataset (web only)	Synthesized set of GUI screenshots and GUI code.	1,750 website screenshots	https://github.com/tonybeltramelli/pix2code/tree/master/datasets
[Yun et al. 2018]	NI	Sketches that have been annotated manually with LabelImg.	50 mimicked sketch images including ~ 600 GUI elements from the screenshot images gathered on the Internet	NI

Table 8: Characteristics of datasets

4.4 Which Kind of ML Technique/Neural Network is Used?

The approaches are quite different in terms of detail and reproducibility based on the presented description of the approaches. Some present a very superficial description, which makes it difficult to identify the applied techniques and parameters employed [Aşıroğlu et al. 2019][Bajammal et al. 2018][Beltramelli 2019][Ge 2019][Halbe and Joshi 2015][Yun et al. 2018]. Although we considered the degree of detail of the description on the ML techniques insufficient for their work to be reproducible, we decided to maintain them for the sake of completeness concerning the mapping.

The task to generate code from a GUI image is divided into several sub-problems in most works, mainly image preprocessing, the detection and classification of GUI elements, and the generation of code representing the GUI. The majority indicates the usage of supervised learning algorithms using as input an image of a GUI as well as typically a representation of the GUI elements, their locations, and dimensions represented as a set of tokens. [Beltramelli 2018] on the other hand uses an unsupervised learning approach presenting the GUI image and context information.

For the detection of GUI elements, most approaches use CNNs, including two-stage detectors, such as Region-based CNN (R-CNN) and its variants (such as Faster R-CNN [Kim et al 2018]) as well as one-stage detectors, such as SSD, RetinaNet [Jain et al. 2019][Pandian and Suleri 2020], YOLO [Yun et al. 2018] and their variants. Several of the approaches that divide the processing into several stages use CNN for mapping the raw input image to a learned representation and then RNN for performing language modeling on the textual description associated with the input picture. A few approaches also unify diverse models into one framework. For example, [Chen et al. 2018] integrate CNN for visual understanding and RNN to encode spatial layout information of CNN features as well as an RNN decoder to generate the target tokens of the GUI framework language.

As a result of the object detection, typically an output representation is given on what is recognized. The GUI representation structure is an object containing the types of the identified components of GUI and their properties.

Reference	Approach	Techniques used	Type of learning	CNN model(s)	Framework	Programming language	Availability of the model
[Aşıroğlu et al. 2019]	hybrid	Filtering and contour detection, custom image classification CNN for feature extraction, with a BiLSTM module for feature analysis	supervised	Custom image classification CNN with BiLSTM	NI	NI	NI
[Bajammal et al. 2018]	classic	Density-based clustering technique	unsupervised	---	NI	NI	NI
Beltramelli 2019]	hybrid	Edge detection, CNN	NI	NI	NI	NI	NI
[Beltramelli 2018]	CNN	CNN (Vision) -> LSTM (Language) -> LSTM (Decoder)	supervised	VGG-based image classification CNN with LSTM-based RNN decoder	Keras	Python	https://github.com/tonybeltramelli/pix2code https://uizard.io/research/#pix2code
[Chen et al. 2018]	CNN	CNN (feature extraction), RNN (spatial layout encoding), RNN (GUI skeleton generation)	supervised	Custom image classification CNN with LSTM-based RNN	Torch	LUA	NI
[Chen et al. 2019a]	hybrid	Edge detection, CNN	NI	NI	NI	NI	NI
[Ge 2019]	CNN	NI	NI	NI	NI	NI	NI
[Halbe and Joshi 2015]	ANN	Learning Vector Quantization Neural Network (LVQ)	hybrid	---	NI	NI	NI
[Han et al. 2018]	CNN	CNN for object detection, CNN for image encoding, and Bi-LSTM for encoding the HTML code.	supervised	CNN+Bi-LSTM, CNN+LSTM+Mask-RCNN, CNN+LSTM+Attention, CNN+Bi-LSTM+Attention+Mask-RCNN	NI	NI	NI
[Huang et al. 2018]	classic	line detection heuristics and Random Forest	---	---	NI	NI	NI
[Jain et al. 2019]	CNN	CNN (object detection)	supervised	SSD (RetinaNet upon ResNet)	cuDNN	C++	NI
[Kim et al. 2018]	hybrid	Edge detection and slope filtering and grouping are used to identify the characteristics of simple components to distinguish layout sections. CNN (object detection)	supervised	Faster R-CNN	NI	NI	NI
[Kumar 2018]	CNN	CNN (feature extraction) and RNN (decoder and predictor)	supervised	VGG16-based image classification CNN with a Gated Recurrent Unit (GRU) RNN as sequence encoder and another GRU as a predictor	Keras	Python	https://github.com/ashnkumar/sketch-code
[Liu et al. 2018b]	CNN	CNN (feature extraction) and RNN (decoder)	supervised	VGG-based image classification CNN with Bi-LSTM-based RNN	Keras	Python	https://github.com/liuyanbinssr/blstm-pix2code

[Moran et al. 2018]	hybrid	Computer vision techniques (GUI elements detection), CNN (GUI-component classification), and kNN (GUI hierarchy assembly)	supervised	Custom image classification CNN with kNN-based GUI container hierarchy assembly	MATLAB	MATLAB	NI
[Pandian et al. 2020b]	CNN	CNN (object detection)	supervised	RetinaNet for Object Detection	NI	NI	NI
[Robinson 2019]	hybrid, CNN	classic: contour detection followed by text detection CNN: semantic segmentation	supervised	hybrid: edge detection with Canny, boundary following with Suzuki, text detection with WTS, and element classification with a custom CNN. CNN: semantic segmentation using an Xception as a backbone for Deeplab v3+	TensorFlow Deeplab	Python	NI
Suleri et al. 2019] [Pandian and Suleri 2020]	CNN	CNN (object detection)	supervised	CNN using TensorFlow Object Detection API. Its detection model is RetinaNet-based (SSD network with ResNet backbone)	TensorFlow Object Detection API	Python	https://api.metamorph.designwitheve.com/docs/ https://github.com/vinothpandian/MetaMorph
[Wallner 2018]	CNN	CNN (feature extraction) and RNN (decoder and predictor)	supervised	VGG16-based image classification CNN with a GRU RNN as sequence encoder and another GRU as a predictor	Keras	Python	https://github.com/emilwallner/Screenshot-to-code/
[Yun et al. 2018]	CNN	CNN [object detection]	supervised	YOLO (version not provided]) for object detection followed by hierarchical grouping (using the nondescript technique)	Darknet	C++	NI

Table 9: Characteristics of ML techniques and neural networks

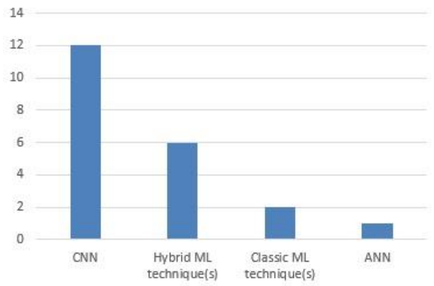


Figure 9: Distribution of ML techniques adopted

Concerning the code generation, most of these methods rely on Domain-Specific Languages (DSLs) that are markup, programming or modeling languages, which are designed for a specialized Domain focusing, e.g., only on relevant information such as the GUI elements and their position, yet, ignoring textual values of labels or visual design details such as color. This representation is then used by a GUI parser to create the code which can be executed on the target platform. With respect to the tools employed, we can observe a clear tendency on the usage of free software tools, with only one approach [Moran et al. 2018] employing the proprietary MATLAB environment. The most used framework was Keras [Chollet 2015], followed by Google's TensorFlow and its extensions. Both frameworks are based upon the Python open-source programming language [van Rossum 1995], making it the preferred tool. Two approaches employ C++ based tools: one uses Nvidia's GPU-based CUDnn API [Chetlur et. al. 2014] and one other work claims to use one of Darknet' YOLO object detection versions. One approach [Chen et al. 2018] employs the LUA script language-based Torch version.

4.5 How Have the Approaches Been Evaluated?

Concerning the evaluation, the researches vary largely in terms of scientific rigor. While some stand out due to their systematic and wide evaluation (including, e.g., [Robinson 2019][Moran et al. 2018] [Chen et al. 2018]), other either present a very superficial evaluation just citing some results without presenting the research design (i.e., [Jain et al. 2019][Kim et al. 2018] [Liu et al. 2018b] [Halbe and Joshi 2015] [Ge 2019]) or do not present any information on the evaluation of the presented approach.

The reported evaluations also vary largely in relation to the analyzed questions and metrics (Figure 10). The majority adopts metrics focusing on the performance of object detection, including mainly accuracy, followed by precision, recall, and F1 value. This stands in contrast to the general indication of mAP as the most adequate performance metric for multi-class object detection [Lie et al. 2020], which is used only in one study [Suleri et al. 2019]. The majority of the studies report accuracy. However, in the context of not uniformly distributed objects (as in the case of GUIs in which, e.g., buttons are much more common than other elements such as toggles), this may introduce biases.

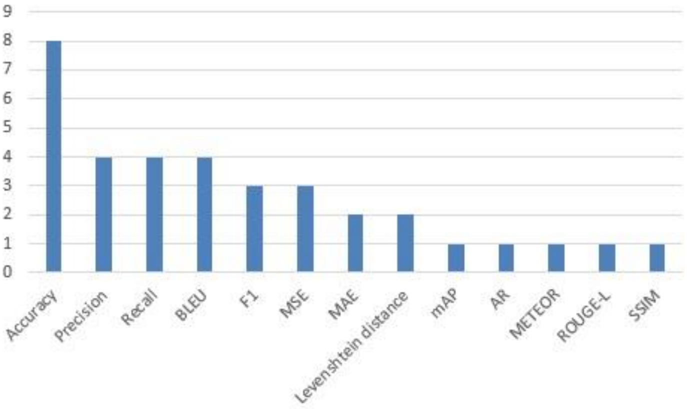


Figure 10: Distribution of usage of metrics

In general, acceptable to good results are presented with, e.g., accuracy varying from 60 to 96%. However, in some cases analyzing, for example, precision per type of GUI element significant differences can be observed ranging, e.g., from 0.562 for paragraphs to 0.896 for images [Robinson 2019].

Several approaches also evaluate the similarity of the generated GUI with respect to the original one, including the visual similarity on pixel level and/or structural similarity by comparing the hierarchical tree similarity ([Chen et al. 2019a][Chen et al. 2018] [Robinson 2019][Han et al. 2018][Moran et al. 2018]). In a few cases similarity has also been evaluated manually either by the researchers [Chen et al. 2018] or GUI designers/developers [Robinson 2019][Chen et al. 2018][Bajammal et al. 2018].

Some studies also compared the Machine Learning approach to classical Computer Vision approaches (such as [Chen et al. 2019a][Moran et al. 2018][Robinson 2019]). As a result, they observed that the Deep Learning approach mostly outperforms other approaches concerning performance. Only in terms of the ability to generalize to previously unseen examples, users rated the classical Computer Vision approach higher [Robinson 2019].

A few studies also analyzed the speed increase caused by the automation varying from 2.5 times [Chen et al. 2018] to 24 times [Beltramelli 2019]. Studies aiming at the analysis of the suitability of the approaches also demonstrated, in general, a good acceptance rate by the participants [Chen et al. 2018][Moran et al. 2018] as well as its usability [Suleri et al. 2019].

Reference	**Testing/evaluation question/measures**	**Sample size**	**Findings**
[Aşıroğlu et al. 2019]	• Method accuracy and validation accuracy	NI	The model achieves 96% method accuracy and 73% validation accuracy.
[Bajammal et al. 2018]	• Correctness of the refactorings of the component generation (precision, recall)	5 expert web developers	VizMod achieves on average 94% precision and 75% recall in terms of agreement with the developers' assessment.

	• Effectiveness in identifying GUI components compared to manual examination by web developers • Amount of code reusability to be achieved through refactoring.		The refactorings yielded 22% code reusability on average.
[Beltramelli 2019]	Speed increase in comparison to the manual design process	20 UI/UX designers and front-end developers	24 times speed increase on average in interface design taking the tool only 170 sec to create high-fidelity prototype and front-end code.
[Beltramelli 2018]	Accuracy	250 GUI images	Pix2code can automatically generate code from a single input image with over 77% accuracy for three different platforms. Minimum classification error is ranging from 11.01% (web) to 22.34% (Android) and 22.73% (iOS).
[Chen et al. 2018]	• Accuracy [percentage of testing pairs whose GUI skeleton exactly match generated GUI skeleton • BLEU: similarity of machine-generated translations and human-created reference translations • Manual study the differences between the generated GUI skeletons and their ground truth • Generalization evaluation • Usefulness evaluation through a user study (time, similarity, and satisfaction)	10,804 GUI images 8 Ph.D. students/ research staff	Accuracy: 60.28% of the generated GUI skeletons exactly match the ground truth GUI skeletons. Accuracy degrades when the GUI skeletons are too simple (≤ 10 GUI components, ≤ 3 containers, and/or ≤ 3 depth). The average BLEU score is 79.09, when the beam width is 1 (i.e., greedy search). The approach can reliably distinguish different types of visual elements and generate the right GUI components. Accuracy and BLEU score are only slightly different when applying on GUI images not included in the dataset demonstrating the generality of the approach. The experiment group implements the skeleton GUIs faster than the control group implementing the design from scratch (with an average of 6.14 min vs. 15.19 min). On average, satisfactory ratings for the experiment and control group are 4.9 versus 3.8, and the similarity ratings for the experiment and control group are 4.2 versus 3.65 are obtained.
[Chen et al. 2019a]	• Accuracy (in comparison with other ML techniques) • Similarity of the generated web pages by mean absolute error (MAE) and mean squared error (MSE). (in comparison with other techniques)	NI	The CNN classification outperforms all baselines (Logistic Regression, SVM, K-nearest Neighbors), achieving more than 85% accuracy, while the baselines achieve 20%-70% accuracy. The GUI similarity of the generated pages with existing techniques, such as PIX2CODE and UI2CODE is between 60%-70%.
[Ge 2019]	The potential of the approach to find visually similar apps	6 sketches	The DL framework is able to generate GUI skeletons for sketches not included in the dataset. Further evaluation is presented only concerning the retrieval of similar apps.
[Halbe and Joshi 2015]	• Accuracy	30 sketches	Preliminary empirical evaluation of the accuracy of results obtained of at least 70%.
[Han et al. 2018]	Comparing different versions of the approach wrt.: • BLEU to compute the co-occurrence frequency of N-grams in candidates and references • ROUGE-L uses the method of LCS (Longest Common Subsequence) • METEOR considers the metric of recall and uses the weighted harmonic mean based on single-precision SUM is the average of BLEU, ROUGE-L, and METEOR	450 sketches	The most complete version of the approach achieves: BLEU 0.679, METEOR 0.513, ROUGE-L 0.783, and SUM 0.658 achieving the same values as reduced versions and/or outperforming them.
[Huang et al. 2018]	• Precision, recall, F1 (per tag e total average) • Accuracy (per tag e total average)	50 web pages	Precision varies from 0.419 (FORM) to 0.911 (SPAN leaves), Recall varies from 0.499 (P) to 0.952 (SPAN leaves), F1 value varies from 0.128 (SPAN inner nodes]) to 0.931 [SPAN leaves]. Accuracy varies from 0.651 (DIV) to 0.964 (SPAN inner nodes).

[Jain et al. 2019]	• Inference time for element detection	50 sketches	The inference time ranges from 0.2 sec to 1.7 sec increasing with the number of elements on a page. Also, images taken at low-light conditions where pixels are darker, increase the inference time.
[Kim et al. 2018]	• Accuracy • Recall	NI	The accuracy is 91% and recall rate 86% of GUI object detection, and it was possible to convert them into HTML code.
[Kumar 2018]	• Similarity of the machine-generated to human-generated GUI using BLEU	NI	BLEU score of 0.76 on the evaluation set
[Liu et al. 2018b]	Accuracy	NI	85% of accuracy about test set.
[Moran et al. 2018]	• Accuracy [Precision] also in comparison to a CV approach • Similarity of the generated GUI hierarchy to original ground truth hierarchies using the Levenshtein edit distance • Visual similarity of generated apps compared to mock-ups calculating the mean squared error (MSE) and mean average error (MAE) across all pixels in screenshots from generated apps for different approaches compared to the original screenshots • Suitability in industrial context based on expert opinion	14,382 GUI images 83 GUI images 3 GUI experts	Precision for the CNN approach is 91.1% outperforming alternative approaches. ReDraw-MockUp produces hierarchies that are closer to the target hierarchies than other approaches (REMAUI and pix2code). REDRAW-CV outperformed both REMAUI and pix2code in MAE, whereas all approaches exhibited very low MSE, with REMAUI very slightly outperforming both ReDraw variants. The results indicate that the apps generated by ReDraw exhibit high visual similarity compared to target screenshots. REDRAW has shown potential for industrial design and development workflows, yet requires to be adjusted to specific processes.
[Pandian et al. 2020b]	NI	NI	NI
[Robinson 2019]	• Precision, Recall, F1 (in comparison with CV) per GUI element • Similarity of the generated website concerning the original website: ○ Visual similarity: structural similarity (SSIM) and mean squared error (MSE) as metrics to evaluate the pixel level visual similarity ○ Structural similarity: Levenshtein edit distance measure comparing the hierarchical tree similarity ○ User evaluation by choosing the best website for a given sketch out of 3 alternatives Generalization of unseen examples through a qualitative user study	250 sketches 22 website experts	The DL approach outperforms the CV approach in element classification by a significantly higher F1 score on the classification of all elements except paragraphs. Precision varies from 0.562 [paragraph] to 0.896 (image), Recall varies from 0.461 (paragraph) to 0.741 (image), F1 score varies from 0.548 (paragraph) to 0.811 (image). Button and input cause confusion for small elements. The DL approach presents a lower MSE and a higher SSIM indicating better performance than the CV approach. The DL approach also demonstrated a lower median concerning edit operation indicating a better performance for structural similarity. Users top rated the DL approach with 22/22 votes regarding similarity, but provided better ratings for the CV approach (15/22) concerning generalization.
[Suleri et al. 2019] [Pandian and Suleri 2020]	• Mean Average Precision • Average Recall • Usability of tool (SUS)	592 sketches 15 UI/UX designers	MetaMorph detects GUI elements from lo-fi sketches with 84.9% mAP with 72.7% AR [Pandian et al. 2020b]. 87% of the participants said that they would like to use Eve frequently to create prototypes. 80% of the users found Eve easy to use. None thought that they would require any technical support to use the system or that the system is unnecessarily complex. The tool scored an average of 78.5 points out of 100, which implies overall good usability.
[Wallner 2018]	Similarity of the machine-generated GUI to human-generated GUI using BLEU	NI	BLEU score of 0.97
[Yun et al. 2018]	NI	NI	NI

Table 10: Characteristics of evaluations

5 Discussion

Considering the importance of the aesthetics and usability of graphical user interfaces and, thus, the importance of the interface design as well as the potential effort reduction by the automatic generation of GUI code, we encountered only a small number of research efforts. While on the other hand, the relevance of the topic itself is also pointed out by the availability of commercial solutions, such as Microsoft AI.Lab's Sketch2Code, among others. With the exception of only one, all articles have been published in the last three years, demonstrating also the recent trend to the adoption of Machine Learning approaches for this task.

We encountered solutions for web GUIs as well as mobile applications, focusing more on Android than iOS. Most approaches are based on sketches as input, recognizing the importance of the creative process of sketching as a first step in prototyping the GUI in contrast to approaches that may aim at complete automation of the GUI design process.

However, we observed large differences between the approaches concerning the types of GUI elements to be detected. Several approaches focus on a rather explorative way only on a very small number of GUI elements. For example, [Aşıroğlu et al. 2019] and [Robinson 2019] only consider four different types of elements, and [Ge 2019] only seven. This limits the applicability of the proposed models in practice, as to be effective, the approaches need to detect all kinds of GUI elements of the respective type of interface. Other researches, such as [Moran et al. 2018], [Suleri et al. 2019], and [Pandian et al. 2020b] already extend the number of GUI elements to 15, 19, and, respectively 25 types of GUI elements. Yet, although several studies justify their selection based on the frequency of usage of GUI elements, some elements such as dropdown, which are still reasonably frequently used, are rarely covered ([Aşıroğlu et al. 2019] [Jain et al. 2019]). We also observed that only [Jain et al. 2019] and [Robinson 2019] differentiated between types of text (including, for example, Heading, Label, and Paragraph).

Research in this area concerning images of GUIs is further complicated due to the unavailability of large datasets such as e.g., ImageNet for images in general. Thus, a considerable effort needs to be spent on the creation of specific datasets as only two researches use pre-existing dataset RICO [Ge 2019] [Pandian et al. 2020b]. And, although, being one of the largest datasets in this field with information on over 9.3k Android apps RICO focuses on the representation of visual, textual, structural, and interactive properties based on screenshots as well as metadata, not providing, for example, sketches. However, as several studies in this field developed their datasets and made them available, especially the pix2code dataset has also been used (partially) by other researches [Kumar 2018][Liu et al. 2018b][Wallner 2018].

Adopting diverse approaches, other studies captured GUI screenshots by typically crawling websites and/or online app stores. In some cases, GUI images or sketches have been synthetically generated. Comparing these to real GUI images and/or sketches, it becomes obvious that there are significant differences concerning the image quality as well as authenticity. However, this issue and its potential impact on performance and validity is not further discussed in most cases. An exception is [Robinson 2019], who reports a reduction of the performance of the deep learning approach when applied to real sketches, emphasizing, thus, the importance of a great variety in sketches to allow the deep learning approach to better generalize to unseen sketching styles.

Most researches were conducted with relatively small datasets ranging from 1,500 GUIs [Han et al. 2018] to about 5,250 GUIs [Beltramelli 2018] with acceptable results, which indicates that ML models for this specific task can be developed with reasonably small datasets, keeping the data preparation effort reasonable.

In terms of ML techniques, we encountered a large variety ranging from classic ML approaches to recent CNNs and diverse combinations. Yet, the majority of the researches adopted CNN.

In order to be able to detect context-related information, such as nested GUI widgets, some approaches employed recurrent convolutional neural network-based techniques from the field of signal and natural language processing, including Gated Recurrent Units, LSTM and Bi-LSTM [Aşıroğlu et al. 2019][Beltramelli 2018][Chen et al. 2018][Han et al. 2018][Wallner 2018].

In terms of evaluation, the large variety of measures indicates a lack of a clear standard for the evaluation of this kind of research. Furthermore, as the large majority focused (in some cases exclusively) on the analysis of accuracy, the evaluation is not necessarily aligned with commonly proposed measures for object detection (such as mAP that has been used only in one research [Suleri et al. 2019]). This lack of a standard for evaluation also hinders the comparison of the approaches as well as future work.

Considering the data presented, in general, acceptable to good results are presented, however, in some cases analyzing, for example, precision per type of GUI element significant differences can be observed. Therefore, the detailed performance results need to be considered carefully, as consequently diverse GUI elements may not be detected with an acceptable degree. As a result of the evaluations some weaknesses have been identified as part of the presented approaches [Robinson 2019] [Moran et al. 2018][Chen et al. 2018][Chen et al. 2019a]:

- Incorrect merging of elements that are close together or when several neighborhood texts in one line use similar fonts and styles
- Incorrect classification of small elements (e.g., misclassifying small button elements as input elements).
- Incorrect implementation of alternative GUI elements, such as e.g., a ToggleButton vs. a Switch for the control.
- Misclassification of ProgressBars and ToggleButtons due to multiple existing styles of the components.
- Degrading accuracy with very simple GUI skeletons (≤ 10 GUI components, ≤ 3 containers, and/or ≤ 3 depth).
- GUI elements that are only partially visible (e.g., covered by a suspension menu), may not be recognized.

On the other hand, a strength observed by [Chen et al. 2018] is the reliable detection of text elements in GUI images even when the texts are written in different languages, making the approach language independent.

In addition, some researches also evaluate the similarity of the code generated adopting typically measures from text analysis, further indicating a lack of specific measures for this kind of research with respect to GUI images. Very few also analyze the applicability of the proposed approaches through user studies [Beltrami 2019][Chen et al. 2018][Robinson 2019][Suleri et al. 2019]. Yet, so far no empirical study on the application of these approaches in practice has been encountered.

Regarding the comparison of the effectiveness of the different approaches, even if Robinson [2019] compared a CNN approach to Computer Vision techniques and demonstrated that the deep learning approach outperforms the CV approach in element classification, but not in container classification, it is impossible to compare the performances reported by the approaches we identified. Regardless of the ML technique used, the approaches are very heterogeneous both regarding the kind of data they employ and also with respect to the techniques they apply for validation studies. Therefore, an objective comparison is not possible as this would require that all approaches employed a common or similar dataset and adopted related evaluation models.

5.1 Threats to validity

As in any systematic mapping, there exist threats to validity of the results presented. Therefore, we identified potential threats and applied mitigation strategies in order to minimize their impact on our research.

Publication bias. Systematic reviews suffer from the common bias that positive outcomes are more likely to be published than negative ones. Nevertheless, we do not consider this an essential threat to our research as rather than focusing on articles that present findings on the impact of these approaches, we aim at eliciting the characteristics of the approaches themselves independent of the evaluation results.

Identification of studies. Another risk is the omission of relevant studies. In order to mitigate this risk, we carefully constructed the search string to be as inclusive as possible considering not only core concepts but also synonyms. The risk of excluding relevant studies is further mitigated by the use of multiple databases, which cover the majority of scientific publications in the field.

Study selection and data extraction. Threats to study selection and data extraction have been mitigated with a detailed definition of the inclusion/exclusion criteria. We defined a rigid protocol for the study selection and all authors conducted the selection together always discussing the selection until consensus was achieved.

6 Conclusion

The approaches we encountered in this mapping study show how Machine Learning can help to facilitate and speed up the design of user interfaces while at the same time maintain initial steps such as sketching as a creative process performed by humans. Yet the small number of researches and the gap to emerging commercial tools indicate the need for further research in this area. Furthermore, the large variety of ML techniques that seems to be employed in a rather explorative fashion as well as the lack of a clear standard for the evaluation of these models, point out important issues to be studied to create conditions for more systematic and comparable research in this field. And, although a few present first user studies with respect to the applicability of the approaches, no scientific evidence on the employment of these approaches in practice has been encountered. Moreover, no indications on how the proposed solutions can be integrated into a more comprehensive prototyping or case tool in order to support the entire process have been found. Thus, as the results show the increased relevance of such support, they also point out the need for further research including the creation of datasets, ML model development as well as their evaluation.

Acknowledgements

This work was supported by CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico – www.cnpq.br*), an entity of the Brazilian government focused on scientific and technological development.

References

[Aafaq et al. 2019] Aafaq, N., Mian, A., Liu, W., Gilani, S. Z., Shah, M.: "Video Description: A Survey of Methods, Datasets and Evaluation Metrics"; ACM Computing Surveys, 52(6) (2019)

[Aşıroğlu et al. 2019] Aşıroğlu, B., Mete, B. R., Yıldız, E., Nalçakan, Y., Sezen, A., Dağtekin, M., Ensari, T.: "Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques"; Proc. of the Scientific Meeting on Electrical-Electronics and Biomedical Engineering and Computer Science, Istanbul, Turkey (2019), 1-4.

[Bajammal et al. 2018] Bajammal, M., Mazinanian, D., Mesbah, A.: "Generating reusable web components from mockups"; Proc. of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France (2018), 601–611

[Beltramelli 2019] Beltramelli, T.: "Hack your design sprint: wireframes to prototype in under 5 minutes"; Medium, (2019) https://uxdesign.cc/hack-you-design-sprints-wireframes-to-prototype-in-under-5-minutes-b7b95c8b2aa2

[Beltramelli 2018] Beltramelli, T.: "pix2code: Generating Code from a Graphical User Interface Screenshot"; Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Paris, France (2018)

[Chen et al. 2018] Chen, C., Su, T., Meng, G., Xing, Z., and Liu, Y.: "From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation"; Proc. of the International Conference on Software Engineering, Gothenburg, Sweden (2018)

[Chen et al. 2019a] Chen, S., Fan, L., Su, T., Ma, L., Liu, Y., Xu, L.: "Automated cross-platform GUI code generation for mobile apps"; Proc. of the 1st Int. Workshop on Artificial Intelligence for Mobile, Hangzhou, China (2019)

[Chen et al. 2019b] Chen, C., Feng, S., Xing, Z., Liu, L., Zhao, S., Wang, J.: "Gallery D.C.: Design Search and Knowledge Discovery through Auto-created GUI Component Gallery"; Proc. of the ACM on Human Computer Interaction, Article 180 (2019)

[Chetlur et al. 2014] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: "cuDNN: Efficient Primitives for Deep Learning" CoRR abs/1410.0759 (2014)

[Cui et al. 2018] Cui, Y., Yang, G., Veit, A., Huang, X., Belongie, S.: "Learning to Evaluate Image Captioning"; arXiv:1806.06422 [cs.CV], 2018.

[Deka et al. 2017] Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: "Rico: A Mobile App Dataset for Building Data-Driven Design Applications"; Proc. of the ACM Symposium on User Interface Software and Technology (2017) 845–854

[Garrett, 2010] Garrett, J. J.: "The Elements of User Experience: User-Centered Design for the Web and Beyond"; Pearson Education (2010)

[Ge, 2019] Ge, X.: "Android GUI search using hand-drawn sketches"; Proc. of the 41st Int. Conference on Software Engineering, Piscataway, USA (2019)

[Han et al. 2018] Han, Y., He, J., Dong, Q.: "CSSSketch2Code: An Automatic Method to Generate Web Pages with CSS Style"; Proc. of the 2nd Int. Conference on Advances in Artificial Intelligence, Barcelona, Spain (2018) 29–35

[Halbe and Joshi 2015] Halbe, A., Joshi, A. R.: "Novel Approach to HTML Page Creation Using Neural Network"; Procedia Computer Science, 45 (2015) 197-204

[Hartson and Pyla 2019] Hartson, R., Pyla, P.: "The UX Book: Process and Guidelines for Ensuring a Quality User Experience"; 2nd Ed., Morgan Kaufmann (2019)

[Ho 1995] Ho, T. K.: "*Random Decision Forests*"; *Proc. of the 3rd Int. Conference on Document Analysis and Recognition, Montreal, Canada (1995) 278–282*

[Huan et al. 2016] Huan, R., Long, Y., Chen, X.: "Automatically Generating Web Page from A Mockup"; Proc. of the 28th Int. Conference on Software Engineering and Knowledge Engineering, Redwood City, USA (2016) 589-594

[Huang et al. 2019] Huang, F., Canny, J. F., Nichols, J.: "Swire: Sketch-based User Interface Retrieval"; Proc. of the CHI Conference on Human Factors in Computing Systems, ACM, New York, USA (2019) 1–10

[Huang et al. 2017] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K. "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors"; Prod. of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA (2017) 3296-3297

[Hechun and. Xiaohong 2019] Hechun, W., Xiaohong, Z.: "Survey of Deep Learning Based Object Detection"; Procs. of the 2nd Int. Conference on Big Data Technologies, ACM, New York, USA (2019) 149–153

[Huang and Canny 2019] Huang, F., Canny, J. F.: **"**Sketchforme: Composing Sketched Scenes from Text Descriptions for Interactive Applications"; Proc. of the 32nd Annual ACM Symposium on User Interface Software and Technology (2019) 209–220

[Jain et al. 2019] Jain, V., Agrawal, P., Banga, S., Kapoor, R., Gulyani, S.: "Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network"; arXiv:1910.08930 [cs.CV] (2019)

[Jiao et al. 2019] Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., Qu, R.: "A Survey of Deep Learning-based Object Detection"; arXiv:1907.09408 [cs.CV] (2019)

[Kilickaya et al. 2017] Kilickaya, M., Erdem, A., Ikizler-Cinbis, N., Erdem, E.: **"**Re-evaluating Automatic Metrics for Image Captioning"; Proc. of the 15th Conference of the European Chapter of the Association for Computational Linguistics, Valencia, Spain (2017)

[Kim et al. 2018] Kim, B., Park, S., Won, T., Heo, J., Kim, B.: "Deep-Learning Based Web UI Automatic Programming"; Proc. of Int. Conference on Research in Adaptive and Convergent Systems, Honolulu, USA (2018)

[Kolthoff 2019] Kolthoff, K.: **"**Automatic generation of graphical user interface prototypes from unrestricted natural language requirements"; Proc. of the 34th IEEE/ACM International Conference on Automated Software Engineering (2019) 1234–1237

[Kohonen 1988] Kohonen, T.: "Learning Vector Quantization", Neural Networks (1988)

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., Hinton, G. E.: “ImageNet classification with deep convolutional neural networks”; Proc. of the 25th Int. Conference on Neural Information Processing Systems (2012) 1097-1105

[Kumar 2018] Kumar, A.: “Automated front-end development using deep learning”; Medium Insight (2018) https://blog.insightdatascience.com/automated-front-end-development-using-deep-learning-3169dd086e82?gi=bcd0cff78213.

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. “Gradient-based learning applied to document recognition”; Proc. of the IEEE, 86(11) (1998), 2278-2324

[Lepore 2010] Lepore, T.: “Sketches and Wireframes and Prototypes! Oh My! Creating Your Own Magical Wizard Experience”; UX Matters (2010) https://www.uxmatters.com/mt/archives/2010/05/sketches-and-wireframes-and-prototypes-oh-my-creating-your-own-magical-wizard-experience.php]

[Liu et al. 2018a] Liu, T. F., Craft, M., Situ, J., Yumer, E., Mech, R., Kumar, R.: “Learning Design Semantics for Mobile Apps”; In Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology, Berlin, Germany (2018) 569-579

[Liu et al. 2018b] Liu, Y.; Hu, Q.; Shu, K.: “Improving pix2code based Bi-directional LSTM”; Proc. of the IEEE Int. Conference on Automation, Electronics and Electrical Engineering, Shenyang, China (2018)

[Liu et al. 2020] Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M.: “Deep Learning for Generic Object Detection: A Survey”; International Journal of Computer Vision, 128 (2020) 261–318

[Memmel and Reiterer 2008] Memmel, T., Reiterer, H.: “Model-Based and Prototyping-Driven User Interface Specification to Support Collaboration and Creativity”; Journal of Universal Computer Science, 14(19) (2008) 3217-3235

[Moran et al. 2018] Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., Poshyvanyk, D.: “Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps”; IEEE Transactions on Software Engineering, 46(2) (2018)

[Nguyen et al. 2018] Nguyen, T. T., Vu, P. M., Pham, H. V., Nguyen, T. T.: “Deep Learning UI Design Patterns of Mobile Apps”; Proc. of the ACM/IEEE 40th International Conference on Software Engineering: New Ideas and Emerging Results (2018)

[Nguyen and Csallner 2015] Nguyen, T. A., Csallner, C.: “Reverse engineering mobile application user interfaces with REMAUI”; Proc. of the 30th IEEE/ACM Int. Conference on Automated Software Engineering, Lincoln, USA (2015) 248-259

[Ozkaya 2019] Ozkaya, I.: “Are DevOps and Automation Our Next Silver Bullet?”; IEEE Software, 36(4) (2019) 3-95

[Pandian and Suleri 2020] Pandian, V. P. S., Suleri, S.: “BlackBox Toolkit: Intelligent Assistance to UI Design”; Proc. of the Workshop on Artificial Intelligence for HCI: A Modern Approach, Honolulu, USA (2020) 25–30

[Pandian et al. 2020a] Pandian, V. P. S., Suleri, S., Jarke, M.: “Syn: Synthetic Dataset for Training UI Element Detector from Lo-Fi Sketches”; Proc. of the 25th International Conference on Intelligent User Interfaces Companion (2020) 79–80

[Pandian et al. 2020b] Pandian, V. P. S., Suleri, S., Jarke, M.: “Blu: What GUIs are made of”; Proc. of the 25th Int. Conference on Intelligent User Interfaces (2020) 81–82

[Petersen et al. 2015] Petersen, K.; Vakkalanka, S.; Kuzniarz, L.: "Guidelines for conducting systematic mapping studies in software engineering: an update"; Information and Software Technology, 64 (2015) 1–18

[Robinson 2019] Robinson, A.: "Sketch2code: Generating a website from a paper mockup"; Dissertation, University of Bristol, UK (2019)

[Schlatter and Levinson 2013] Schlatter, T., Levinson, D.: "Visual usability: Principles and practices for designing digital applications"; Morgan Kaufmann (2013)

[Sethi et al. 2019] Sethi, N., Kumar, A., Swami, R.: "Automated web development: theme detection and code generation using Mix-NLP"; Proc. of the 3rd Int. Conference on Advanced Informatics for Computing Research, Shimla, India (2019) 1–6

[Sherstinsky 2020] Sherstinsky, A.: "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network"; Physica D: Nonlinear Phenomena, 404 (2020)

[Silva da Silva et al. 2011] Silva da Silva, T., Martin, A., Maurer, F., Silveira, M.: "User-Centered Design and Agile Methods: A Systematic Review"; Proc. of the Agile Conference. (2011) 77-86.

[Sokolova and Lapalme 2009] Sokolova, M., Lapalme, G.: "A systematic analysis of performance measures for classification tasks"; Information Processing and Management, 45 (2009) 427–437

[Russell and Norvig 2010] Russell, S.J., Norvig, P.: "Artificial Intelligence: A Modern Approach"; 3rd Edition, Prentice Hall (2010)

[Suleri et al. 2019] Suleri, S., Pandian, V. P. S., Shishkovets, S., Jarke, M.: Eve: A Sketch-based Software Prototyping Workbench"; Proc. of the Conference on Human Factors in Computing Systems Extended Abstracts, Glasgow, UK (2019)

[Xiao et al. 2019] Xiao, X., Wang, X., Cao, Z., Wang, H., Gao, P.: "IconIntent: Automatic Identification of Sensitive UI Widgets Based on Icon Classification for Android Apps"; Proc. of IEEE/ACM 41st Int. Conference on Software Engineering, Montreal, Canada (2019) 257-268

[Yun et al. 2018] Yun, Y; Jung, J.; Eun, S.; So, S.; Heo, J.: "Detection of GUI elements on sketch images using object detector based on deep neural networks"; Proc. of the 6th Int. Conference on Green and Human Information Technology, Chiang Mai, Thailand (2018)

[Zou et al. 2019] Zou, Z., Shi, Z., Guo, Y., Ye, J.: "Object Detection in 20 Years: A Survey" ; arXiv:1905.05055v2 [cs.CV] (2019)

[Wu et al. 2020] Wu, X., Sahoo, D., Hoi, S. C. H.: **"**Recent Advances in Deep Learning for Object Detection"; Neurocomputing, (2020)

[Wallner 2018] Wallner, E.: "Turning Design Mockups into Code with Deep Learning"; Floydhub (2018) Article available at: https://blog.floydhub.com/turning-design-mockups-into-code-with-deep-learning

[Wang et al. 2004] Wang, Z., Bovik, A., Sheik, H.R., Simoncelli, E.P.: "Image quality assessment: From error visibility to structural similarity"; IEEE Trans. Image Process 4, 13(4) (2004) 1–14

[Williams and Zipser 1989] Williams, R. J., Zipser, D. A "Learning Algorithm for Continually Running Fully Recurrent Neural Networks"; Neural Computation, 1:2 (1989) 270-280

[Yujian and Bo 2007] Yujian, L., Bo, L.: "A normalized Levenshtein distance metric"; IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(6) (2007) 1091–1095

[Zhang 2016] Zhang Z.: “Introduction to machine learning: k-nearest neighbors”; Annals of translational medicine, 4(11), (2016)

[Zhao et al. 2019] Zhao, Z.-Q., Zheng, P., Xu, S.-t., Wu, X.: “Object Detection with Deep Learning: A Review”; IEEE Transactions on Neural Networks and Learning Systems, 30(11) (2019)