LMU Munich
Frauenlobstraße 7a
D–80337 München
Institut für Informatik

Master Thesis

Intent Prediction with Vectorized Sequential Android UI Tree Data

August Oberhauser

august.oberhauser@campus.lmu.de

Course of Study:	Medieninformatik
Examiner:	Prof. Dr. Sven Mayer
Supervisor:	Florian Bemmann, M.Sc.

Commenced:	June 20, 2023
Completed:	January 8, 2024

Abstract

The intent of a user is a source of information, which is hard to accommodate. In the context of using a smartphone, this can imply the next user action, the user flow, or the recommendation for the next app. A flexible, open-source proof-of-concept for an intent prediction model based on a Long Short-Term Memory (LSTM) has been worked out. It includes three models ClickOnly, SelectedFeatures and FeaturesClickShifted, which aim to predict the next user click. All three are performing only slightly better than the statistically mean click gesture, trained on 15% of the usable Rico dataset. Criteria for a dataset suitable for Android intent prediction were determined. Approaches, like transformer multi-attention model, actionable element prediction, and further metrics have potential to push the development of semantically meaningful prediction models.

Contents

List of Figures

List of Tables

Glossary

accuracy Probability of correctness of a classification. 30, 31

big data Extremely large and complex data sets which can only be processed with modern computing soft- and hardware. 20

gradient descent Optimization algorithm that minimzes the error in a differentiable function. 23, 24

one-hot A combination of multiple binary values, where one of them is **1** and the others are **0**. 21, 22, 23, 24

protocol buffers A language-neutral, platform-neutral extensible mechanisms for serializing structured data, developed by Google, see https://protobuf.dev. 30

rooting Method to gain privileged access to the operating system Android. 30

transformer Uses a multi-head attention mechanism, which enables parallel processing for faster computing times. 42

Acronyms

Adaptive Moment Estimation (Adam) A stochastical utility based on adaptive gradiend descent to optimize a function. 39

Application Programming Interface (API) . 17

Artificial Neural Network (ANN) . 22, 23, 24, 27

Autoencoder (AE) . 38

Continuous Bag-of-Words (CBOW) . 28

Convolutional Neural Network (CNN) . 28, 30

Gated Recurrent Unit (GRU) . 41, 42

Global Positioning System (GPS) . 36

Graphical User Interface (GUI) . 28

Graphics Processing Unit (GPU) . 34, 37, 41

Graph Neural Network (GNN) . 28

Integrated Development Environment (IDE) . 34

Long Short-Term Memory (LSTM) . 2, 31, 37, 38, 41, 42, 43

Machine Learning (ML) Scientific approach to form statistical models without the need to explicitly program it. 14, 20, 21, 22, 33, 34, 41, 43

Mean Squared Error (MSE) . 24, 39, 40

Mean Absolute Error (MAE) . 24

Neural Network (NN) . 23, 28, 30, 34, 35, 43

Operating System (OS) . 13, 17, 19, 30, 33, 36, 41

Recurrent Neural Network (RNN) . 22, 23, 30, 38, 41, 42

Residual Neural Network (ResNet) . 28

Root Mean Squared Error (RMSE) . 23, 24, 33, 39, 41

Social Networking Service (SNS) . 31

Unidirectional Data Flow (UDF) . 17, 18

User Interface (UI) . 14, 17, 18, 19, 21, 30, 42

1 Introduction

The interaction of a user with an end device such as a smartphone or a computer is very diverse and difficult to contextualize. Nevertheless, user-specific (personalized) as well as global (collaborative) patterns can possibly be worked out with the help of preceding user interactions and screen contents. These could be used to predict the intention, such as the action or the motivation of a user or a group of users. Especially the level of detail, at which these predictions can be made, is an integral part of offering a reliable service. By making use of the continuous on-device data, the users behavior can be worked out and be used to forecast their next actions. These can then be very coarse, such as predicting the next app. Or they can be very detailed, e.g. determining the next user action, such as filling out a form field or selecting the favorite meal in the delivery service app.

1.1 The Role of Intent Prediction

The term *intent* can have ambiguous meanings and can be used in different context. In the dictionary it is described as the fact of intending something, so it is planned to do [13]. Generally we can assume that intent is a stronger desire to accomplish ones intention. Kofler and Hanjalic et al. [20] also describe the intent as "immediate reason, purpose, or goal [. . .] that motivates a user to" act.

The aim or purpose must be differentiated from the actual user input, also known as interaction. So the intent can be seen as the preliminary stage of interaction. Gestures and click sequences then are the concrete actions, and might fulfill a (small) part of the users intent. Therefore, in this work it is not the goal to obtain the users full intent, but to work out factors, such as user inputs or gestures, which hint to the intent of the user. Further, *prediction* describes that the intent or factors of intent should be available (calculated) before they have actually happened or were measured.

1.2 Necessity of Vectors for Android UI

A vector is a mathematical construct, which can hold a list of numbers, and it is also a special form of a tensor. Vectorizing an object means, representing the object in a concise form and ensuring to transfer certain features or meanings. In this context, vectorization is a tool to embed objects in a (small) binary package, which then can perform certain actions in an artificial environment. Most of the time in this thesis vectorization is referred to as embedding or the result of a trained model.

The Android Operating System (OS) produces a big amount of data in every user session. To be able to process such an amount of these traces, it is helpful to save these in a small form factor, such as an embedding.

Furthermore, such a technology provides many benefits in addition to intent prediction. The complexity and the size of a User Interface (UI) tree can be reduced by vectorizing them. User groups can be worked out in social studies, which have a similar behavior when using digital UI systems [18]. Also, the technical expertise on individual features can be eliminated, that would be required to manually compare user sessions [16]. A users smartphone usage history can be saved or compared, considering the aspect of avoiding saving personal information. App developers can be supported to improve their app design and usability. It may be applied in psychology and market research or help detecting addictions. On a technical side the method can optimize preloading of processes on mobile devices, which results in energy savings [30]. As shown, many fields of application can profit by elaborating such a system. It would be exciting to know, how the concrete concept would look like and if it can be implemented successfully e.g. to improve the user experience on end-user devices.

For this purpose, the Android UI of the device can be tracked, and relevant information are collected. After a few preprocessing steps, the data can be trained with a Machine Learning (ML) model to acquire the user behaviors and then make predictions, if convenient for the user.

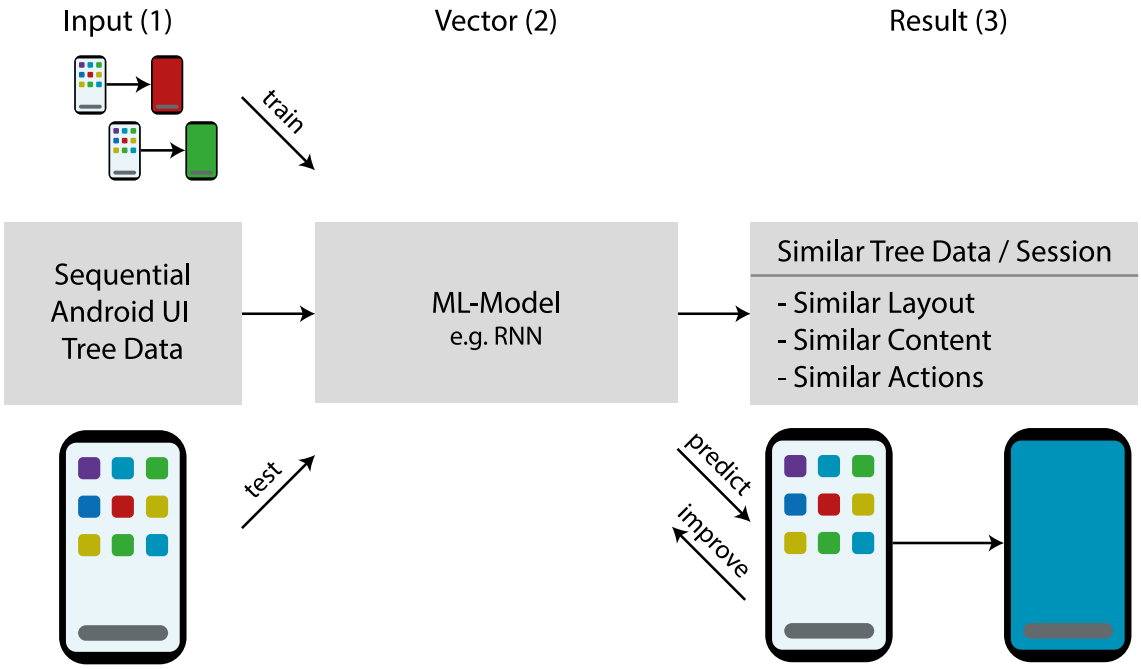


Figure 1.1: Possible procedure using a Machine-Learning algorithm to predict the next intent from a beginning user session: The input (1) can be a sequence of Android tree data. With help of a Machine-Learning-Model (2) (e.g. RNN) a vector representation (embedding) can be trained and then predict the most probable action or screen (3) from a given starting sequence, but also can be improved through the users feedback with reinforcement learning.

A guideline and a proof-of concept will be developed on how a model for predicting user intent could be built. To this end, possibilities for collecting and vectorizing sequential UI trees (e.g., from the Android Accessibility Service) will be discussed. The performance of the model can be measured, for example, by indicators such as the amount of training data and time spent on the learning process.

1.3 Contributions

The thesis is aimed to contribute in the following way. Criteria on dataset for Android UI embeddings are given. A proof-of-concept is shown, how a multidimensional flexible intent prediction model can be designed and, an overview of methods and models is given, which can help deciding which model is suited for Android intention prediction.

In computer science there often coexists multiple (correct) solutions to the same problem. Many technologies have the capabilities of achieving a similar result, but are specialized in the one or the other way. That led to the question of *what a suitable model for the prediction of user intent* can be.

Also, the term “user intent” is not defined exactly. A prediction can be made in different contexts. That raises the question *at what level of detail the intent predictions can be made?*

2 Theoretical Framework

Before starting with rehabilitate the main topic, some basic concepts, technical terms and underlying theory has to be explained. This is important to faster read through the thesis without the need to interrupt the flow of thought. In the following, a set of established theories and terms is explained which are not directly related to the thesis formulation, but essential to comprehend numerous contexts.

2.1 Android UI

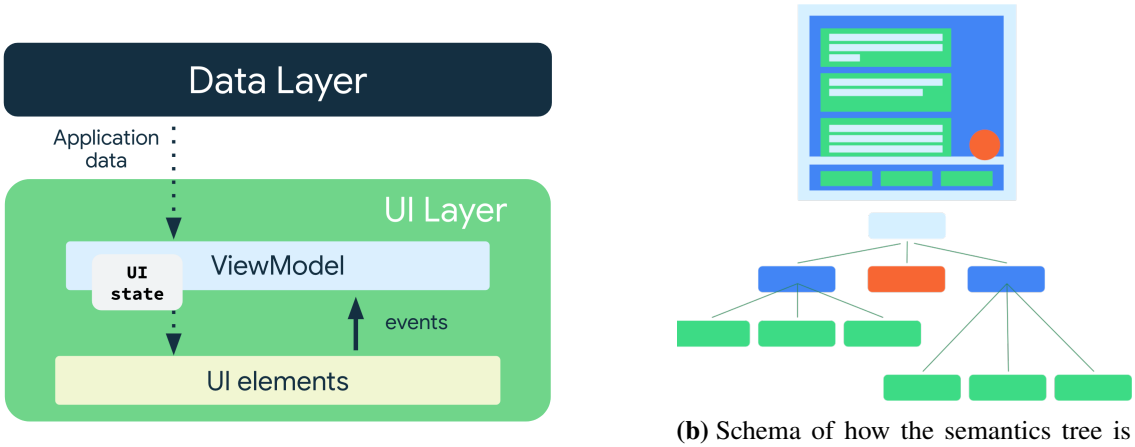
To be able to use any of the things displayed on the mobile device, some concepts of UI programming must be shown. A UI enables the user to view the applications data on the screen but also to interact with the device especially on mobile devices [11].

The main challenge is to bring the application data in the right format, so that the display can interpret the instructions to draw the elements. Each mobile phone with the Android OS has a basic set of native functions through which the UI can be drawn and updated, a so-called Application Programming Interface (API). These functions can be very general to instruct drawing a whole component such as an alert box, or they can be very specific as drawing single rectangles in a canvas.

The rough transition from the application data (data layer) to the display data (ui layer) is depicted in figure 2.1a. The application data is transformed, concatenated or filtered to be saved in a view model, which represents the state for each view. The view model is then layed out to multiple UI elements. E.g. they are loaded in the Android activity via view layouts [8] or composed in a declarative approach [10]. It is generally advised to use a Unidirectional Data Flow (UDF). This ensures that the data is only changed in one place and doesn't get out of sync between UI elements, the view model and the data layer. The UI can also register user inputs (like a button press) and report them back to the view model. The view model then updates the application data, if needed, and then also reports the current UI state back to the UI elements to be rerendered.

2.1.1 Data Tree Structure

The UI elements (i.e. the composition) themselves are hierarchically structured in a tree. This allows the renderer to calculate relative distances, floatings and skip processing hidden or overlapped elements.



(a) Diagram of UDF in app architecture [11]

(b) Schema of how the semantics tree is related to the UI hierarchy [9]

Figure 2.1: Structure of the Android UI

With the Android Layout Inspector (figure 2.2) a view hierarchy tree can be visually inspected while displaying its position and layout on the Android screen. Also, the layout attributes can be validated. This tool allows to debug complex UIs especially when using nested components and display them in a simplistic way. Note that this tool is only available if one has access to the app's source code.

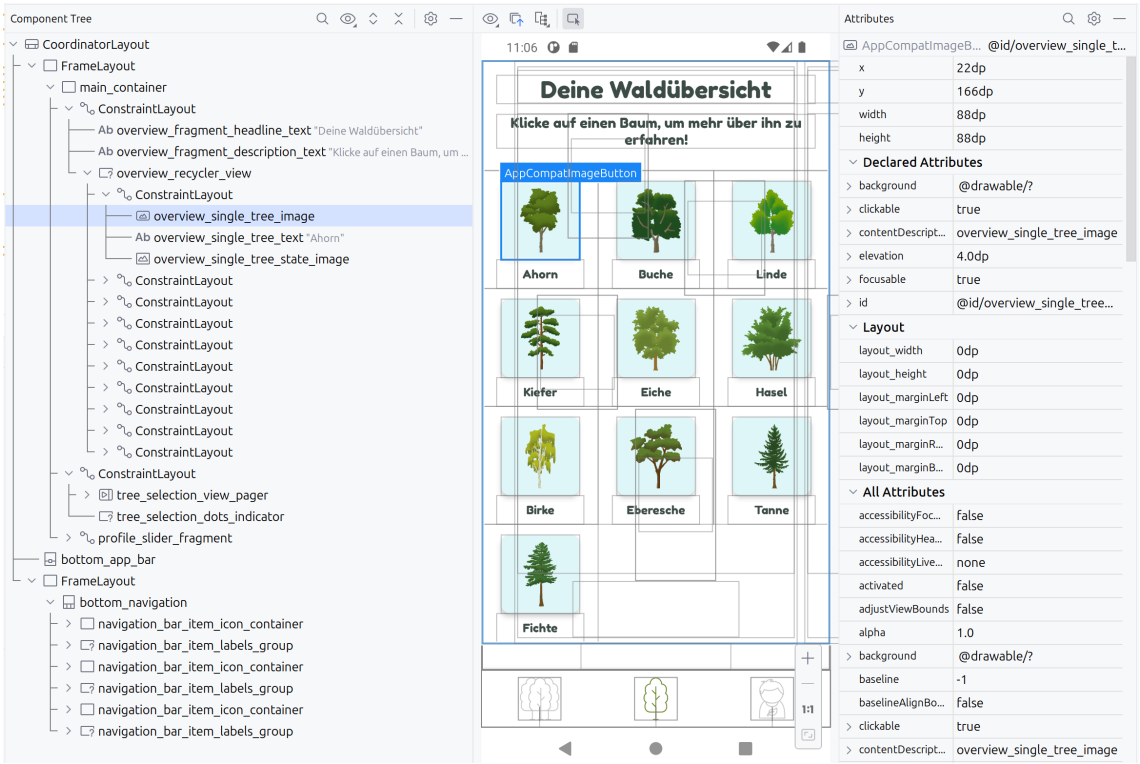


Figure 2.2: The Android Layout Inspector tool using the example of "App ins Grüne" [1] by the Media Informatics Group of the LMU in Munich.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<hierarchy>
    <node index="0" text="" resource-id="" class="android.view.ViewGroup"
          package="de.lmu.treeapp" content-desc="" checkable="false"
          checked="false" clickable="false" enabled="true" focusable="false"
          focused="false" scrollable="false" long-clickable="false"
          password="false" selected="false" visible-to-user="true"
          bounds="[0,137][1440,2923]">
        <node index="0" text="" resource-id=""
              class="androidx.viewpager.widget.ViewPager" package="de.lmu.treeapp"
              content-desc="" checkable="false" checked="false" clickable="false"
              enabled="true" focusable="true" focused="false" scrollable="true"
              long-clickable="false" password="false" selected="false"
              visible-to-user="true" bounds="[4,141][1436,2707]">
            <node index="22" text="Eiche" resource-id=""
                  class="android.widget.TextView" package="de.lmu.treeapp" ... />
        </node>
        <node index="1" text="" resource-id="" class="android.view.ViewGroup" ... >
            <node NAF="true" index="0" text="" resource-id=""
                  class="android.widget.FrameLayout" package="de.lmu.treeapp" ... />
        </node>
    </node>
</hierarchy>
```

Listing 2.1: Android Accessibility Node in XML.

2.1.2 Android Accessibility Service

If an app is running in production on a users device, meaning that the app is compiled and publicly available, the ways of accessing the Android UI tree are limited. This behavior is of course wanted for safety and privacy reasons. Nonetheless, if desired, a user can explicitly allow certain apps to gain access to the semantics tree of your Android OS. This is especially useful for providing accessibility services for impaired users (like done with the TalkBack app). Or, as in our case, the setting can be used to enable services which collect data for user studies or scientific experiments.

This semantics tree is deviated from the existing UI tree. It can be fed via special semantic properties while composing the UI, e.g. by specifying the `contentDescription` property of an icon [9]. Providing semantics is not limited to native platforms as shown by Flutter [3] and React Native [12]. In figure 2.1b a schema is presented, which shows how the elements of the semantics tree are spanned compared to the components on the UI layer.

To take advantage of the semantics tree, a custom accessibility service can be built, which can run in the background. This service tracks all UI changes and has access to the current view hierarchy of the screen, which also inherits the semantic tree. By altering the code of the *AccessibilityNodeInfoDumper* one can extract the view hierarchy to a locale or remote database [29]. In the code section 2.1 a small fraction of a view hierarchy is shown. It contains nested nodes with various attributes which represent the components of the combined UI and semantics hierarchy.

2.2 Machine Learning

ML, a term spread by Arthur Lee Samuel, is a method of data analysis, more precisely a scientific approach to form statistical models without the need to explicitly program it [26]. It uses algorithms to iteratively learn how data is structured. In contrast to statistical inference or manually crafted statistical models respectively, ML can solve tasks by automation of model building. Its advantages lie in finding hidden relations and patterns from the context, without having any or only a small pre knowledge of the data, thus it is a strong tool for generalization or abstraction of large datasets, also known as Big data. ML can be applied to the following fields among others: email and spam filtering, fraud detection, cybersecurity, web search engines, recommender systems (like known from Netflix or Amazon), advertising, translators and text generation, pattern and image recognition. The data driven approach also comes with some drawbacks: the outcome heavily depends on the provided data. It can include biases and therefore may acquire forms of discrimination or unfair treatment. Nonetheless ML has a lot of potential to uncover hidden connections in large datasets. The most important concepts are explained to be able to follow certain decisions in this thesis.

2.2.1 Preprocessing

Preprocessing describes the step after one acquired their data, but before training the ML model. This step is not to be underestimated. A ML model can perform significantly better when certain preprocessing steps are applied [2].

To be able to preprocess, we have to know with what kind of data we handle with. Data entries can occur in different forms, but we can break them down in three main types:

- **Categorical values**: a value is always assigned to a class with a fixed pool of predetermined classes. E.g. letters, words, brands, animals, chemical elements
- **Continuous values**: the value can be fractional and may lies in between a lower and an upper bound. E.g. temperature, velocity, geographic position
- **Integer values**: the value is a whole number and may also lie in between a lower and an upper bound. E.g. revolutions per minute, product number, annual sales

For all types – discrete, continuous, and categorical values – we have a wide variety of options for preparing them in order to be subsequently processed by a ML model [14]. These are listed in the following, while not all can be applied to all types of values.

2.2.1.1 Feature Selection

Feature selection is a crucial step to successfully develop a model with the desired results. It can improve learning performance, thus reduces time, increases computational efficiency, decreases memory storage, and helps build better generalization models [23]. Also, it may be a valid approach to get around missing data and can help structure the data by removing unnecessary clutter.

On the technical sight, feature selection can be differentiated for two goals: supervised and unsupervised learning [23] (cf. section 2.2.2). However, principles from the supervised feature selection can be applied in the unsupervised domain, resulting in a semi-supervised filter selection.

For classifiers and regression problems (unsupervised) following methods can be applied. Multiple features can be compared by calculating their correlation. If one feature is uncorrelated to all other features, this may be an indicator that this feature can be dropped, as it doesn't contribute to the resulting model (*filter method*). However, this can only be stated for linear correlations, thus it can contribute to the result in an unpredicted way. Also, if two features correlate too much to each other, one feature probably is redundant and can be dropped. A good approach is also to reduce the dimensionality of the input data, e.g. by replacing one-hot encoded features of the same domain with embeddings (*embedded method*). This is also called *feature extraction*. Feature extraction can also be used in unsupervised feature selection. Clustering is a common approach to reduce the number of input dimensions by gaining insights of which classes can be merged and which need to stay. A more computational but promising solution is to filter the features by optimizing the model result, also called a *wrapper method*. By gradually removing and adding features and calculating the models performance, one can determine which inputs are important and which are at risk to increase computing time without noticeable effect.

2.2.1.2 Missing Data

Some data entries may are missing. Therefore, two approaches can be employed to get around these missing values. First, one can drop these values by removing the column or row. This is only recommended if you are not relying on this data entry, or this the whole feature is not expected to be important enough to bring any value to the model's performance. Second, the data can be filled with a default value like zero or calculate a reasonable value from the surrounding data entries by taking their "mean, median, or interpolation" [14].

2.2.1.3 Normalization and Standardization

Many ML models work better or exclusively with normalized data. This means that the values have to be in a certain range, most commonly are from **0** to **1** or from **-1** to **1**. This can be achieved by dividing all values with the difference of the minimum and maximum value and shift the output accordingly [14].

Sometimes this is not enough, e.g. if having a few extreme values, and an approach is desired which better reflects the average data. Here the standardization, also called z-score normalization, is applied. This method scales the values so that the mean value is placed at **0** and the standard deviation is placed at **1**.

2.2.1.4 Padding

Padding is a technique to adapt input sequences or matrices of different dimensions to the same size. For example if one screen only has five UI elements, and the next one has twenty, the input size varies significantly. Therefore, there exists three approaches to overcome this problem [27]. The first one is to extend all inputs of a specific feature to the longest available input dimension of this feature. Then every sequence has to be filled with additional values, almost always **0** is used for that, until it matches the longest dimension (*same padding*). Another technique called *valid padding* cuts all data values after reaching the smallest dimension of all inputs of the feature. One

can also use a combination to e.g. pad all inputs with zero to the mean dimension, but throw away all exceeding values as they aren't expected to contribute to a better result. *Causal padding* is a form of *same padding*, but it prepends the fill values in front of the sequence. This is helpful if having Recurrent Neural Network (RNN)s, which rely on a lengthier inputs in order to predict their next value, without need of filtering out short sequences.

2.2.1.5 One-Hot-Encoding for Categorical Values

A categorical value must be treated differently than numerical ordinal values. This is demonstrated best on a concrete example. Imagine a dataset with pictures of animals, and we want to categorize their species. Then the values of all possible species like `dog` and `cat` is called *vocabulary*. To write it in a table, one could add a column called `species` and write their species as a *string* (cf. table 2.1a). Unfortunately a Artificial Neural Network (ANN) cannot work with *strings*, but only with numerical values. So one could think that mapping each species to a number can solve this issue. Indeed, this is possible for ordinal categories, which have a strict linear order, such as ratings or gradings. They are then treated the same as **integer values**. But animals aren't structured ordinal, thus each species must be treated equally. To reflect that we split the species column in multiple sub-columns, so that each new one represents one species, see table 2.1b. If the animal belongs to that species, one inserts a **1** or **True** and if not, a **0** or **False** is filled. This is also called one-hot-encoding [4].

I_d	**Img**	**Species**
0	*blob*	cat
1	*blob*	dog
2	*blob*	cat
3	*blob*	horse

(a) Raw data table

I_d	**Img**	**Cat**	**Dog**	**Horse**
0	*blob*	1	0	0
1	*blob*	0	1	0
2	*blob*	1	0	0
3	*blob*	0	0	1

(b) One-hot encoded species

Table 2.1: One-hot-encoding of categories, I_d is the index of the data entry

2.2.2 Types of Machine Learning

To get be able to evaluate which subfield of ML is the best choice, we need an overview of the existing approaches listed in [15].

Supervised Learning is used for *Classification* and *Regression* tasks. It uses *labeled* samples, which means the input x and output y is known beforehand. Essentially it learns through comparing the actual output (labels), which is provided, with the output the model *predicts*.

Unlike in Supervised Learning, **Unsupervised Learning** isn't provided any labels y. It is used for clustering, dimensionality reduction or anomaly detection. E.g. in clustering the ML model detects similar objects and groups them together. What is seen as similar is up to the model, so it is not known before, with what groups or clusters the model comes up with.

The algorithms using **Semi-supervised Learning** seek to adopt from unlabeled and labeled data [32]. E.g. for a classification problem, the model is either trained first with unlabeled data resulting in clusters, which are then provided a class with a few labeled samples. Or classes are trained based on labeled data, but their class boundaries are extended through unlabeled data, which follows the approach of clustering.

Reinforcement Learning is closely related to decision theory [15]. A model is given the input and produces an output, which is exposed to the environment. The environment then provides a reward or a punishment to the next learning iteration of the model. This then tries to maximise the rewards and minimize the punishments. This learning type can also be helpful in fine-tuning RNNs.

2.3 Artificial Neural Nets

Unfortunately the concepts of neural networks, cannot be explained in detail, as it is beyond the scope of this thesis. A good choice to make familiar with Neural nets is the book "Neural Networks and Deep Learning" of Nielsen [28], which is consulted throughout this section.

An ANN uses biological neuron systems as paradigm to generate mathematical models. The biological neuron model was transferred to computer science and is called *perceptron model*.

A perceptron – or also named neuron in our context – is fed by one or multiple inputs x_i and performs a function f on them resulting in an output $\hat{y}$, where i is an index of the n input values. Also, a weight w_i and a bias b_i can be applied to each of these inputs.

$$\hat{y} = \sum_{i=0}^{n} x_i w_i + b_i$$

Multiple of these neurons can be used together producing one or multiple outputs, thus forming a neural layer, which can act as Neural Network (NN) together with the input and the output layer. If multiple neural layers (also called *hidden layers*) are connected to each other, it is called a *deep neural net*. The output(s) $\hat{y}$ of one or multiple neurons then serve as the input(s) x of one or multiple neurons in the next layer. The last layer – the output layer – then serves the values, which have to be predicted. These values can be compared to the real data (labels) using an error function like Root Mean Squared Error (RMSE). The whole model can be seen as a cost function, which has to be minimized during training. In the training process weights w and biases b for each neuron are adapted, in order to produce outputs, which matches best the labels. This is done iteratively using an optimizer (Adaptive) Gradient descent and *backpropagation*, by searching the minimum cost in multiple steps, called *epochs*.

2.3.1 Embedding Layer

As mentioned in section 2.2.1.5 categories can be represented using one-hot encoding. The conversion can result in innumerable amount of columns, which is equivalent each having its own feature dimension. To reduce the dimensionality of such encoding, so-called categorical *embeddings* can be introduced [4]. This means that each single species is represented by a vector. The length of the vector can be selected freely, it is called the *embedding dimension*. A lookup-table is created which encodes each category with randomly initialized weights of size of the embedding dimension

(cf. table 2.2b). Now the embedding vector for each species is looked up and replaced in the data table resulting in an embedded representation (cf. table 2.2a). During training of the model the weights of the look-up table are successively updated (like for the dense layer) to reduce the error (loss) of the overall model. An embedding would result in an one-hot-encoding, if the embedding dimension and the vocabulary size are equal and the conversion matrix (lookup-table) is the identity matrix.

I_d	**Img**	**SP_1**	**SP_2**
0	*blob*	0.1	0.6
1	*blob*	0.4	0.8
2	*blob*	0.1	0.6
3	*blob*	0.5	0.5

(a) Species encoded with embedding

Species	I_s	**SP_1**	**SP_2**
0	cat	0.1	0.6
1	dog	0.4	0.8
2	horse	0.5	0.5

(b) Lookup table for the 2-dimensional embedding of species

Table 2.2: Embedding of categories, I_d is the index of the data entry, I_s is the index of the species

2.4 Evaluation and Metrics

Accuracy, precision and recall, as well as the combined F_1-Score is only applicable for categorical labels, thus classification.

For regression tasks, what we are aiming for, the Mean Absolute Error (MAE), the Mean Squared Error (MSE) and the RMSE are suited much better.

$$\text{RMSE}(y, \hat{y}) = \sqrt{\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2 / N}$$

with the number of samples n, the actual value y, and the predicted value $\hat{y}$ [17].

The MAE is the mean value when summing all absolute errors $y_i - \hat{y}_i$ and divide them by n The MSE squares the single absolute errors in the MAE to punish single outliers. The RMSE is the root of the MSE to avoid squaring units, but also provide a realistic error values over all samples.

As *loss* is meant the cost or the error, which is differentiating the label from the prediction. In most cases the loss function is the MSE which must be optimized, e.g. with Gradient descent.

As explained in 2.3 ANNs can be trained in epochs. If a model was trained too few epochs, it probably is not able to grab the general trend or learn enough about the data to built context. This is called **Underfitting**. The opposite **Overfitting** is the case, if model was trained too many epochs. This results in knowing the train dataset very well and being able to predict every sample in the training set. But as soon as it is confronted with new data samples, it results in a larger errors than in earlier epochs. This can be solved by adding an *early stop* function, which automatically interrupts the training, when the validation loss increases.

3 Related Work

In the following the most important works related to Android UIs and intent prediction are listed.

3.1 Datasets of UI Trees

A fundamental part of training ANNs are solid and large datasets, which provide enough information to gain the expected result. Therefore, these datasets have to fulfill some requirements, like completeness, quantity, feature richness, topicality and (public) accessibility. In the follows, papers of thematic relevance are verified on the basis of these criteria.

3.1.1 ERICA

ERICA is a design and interaction mining application, which allows gathering *interaction traces* by capturing the users activity on Android apps [6]. This is accomplished through a web-based interaction layer in contrast to the other common approach of using *accessibility services* directly. They justify that approach by the lack of need to install additional applications, as only a browser is required. A further reason is the response latency of the commonly used *UiAutomator*, which cannot collect the data in time. Also they argue that capturing and simultaneously interacting with the apps may overload the user device and challenges the user experience. Therefore the much more powerful servers take the task of capturing the UI trees. The apps are hosted on multiple physical devices with a modified Android OS directly connected with the server. ERICA captures UI screens and user flows by tracking UI changes. They then used this data to form k-mean clusters from the UI elements (visual and textual features) and the interactive elements (icons and buttons). Based on the clusters they then build classifiers and trained an AutoEncoder to determine the flows from the test dataset. The authors worked out 23 common user flows (from over a thousand popular Android apps) which aim to provide complementary, promising or new design patterns and trends.

3.1.2 Rico / Rico-SCA

Rico [5] is the successor of ERICA. It aims to help perform better at designing and support the creation of adaptive UIs. As far as known to date this is the largest collection of mobile app designs and traces with covering 72k UI screens in 9.7k Android apps. Like its predecessor Rico uses a web-based approach to collect user traces. It enables the applications like searching for designs, generation of UI layouts and code, modeling of user interactions, and prediction of user perception. It exposes visual, textual, structural, and interactive design properties of more than 72k unique UI screens. The traces dataset is categorized by apps, wich each inherits multiple interaction traces. These contain a screenshot and a view hierarchy for each screen change. Unfortunately the dataset

doesn't include interaction traces for app to app transitions or interactions with the Android OS itself. In table 3.1 a collection of all view hierarchy attributes is shown with their meaning. These were extracted by iterating over all view hierarchy files contained in the traces of the dataset. This gives insights in what attributes were recorded in the Rico dataset and what relevance they may have during training the model. The authors of Rico used their dataset to train a 64-dimensional UI layout vector 2.3.1 with an AutoEncoder. For their input they converted the UI layout hierarchy to an image with colored bounding boxes differentiating images and text. This has the advantage to be able to deal with the high dimensions inside the UI tree. But the conversion also most likely discards lots of meaningful information hidden in the UI tree semantics.

The Rico-SCA dataset has been formed out of the research topic of mapping language instructions to mobile UI action sequences [25]. They removed screens whose bounding boxes in the view hierarchies are inconsistent with the screenshots with the help of annotators. The process of filtering reduced the Rico dataset to 25k more concise and meaningful screens.

3.1.3 Mobile UI CLAY Dataset

The Google researchers Gang Li et al. [22] present a a so-called *CLAY* pipeline which is able to denoise mobile UI layouts from incorrect nodes or adding further semantics to it. As basis they used the Rico 3.1.2 dataset for a subject of improvement. They state that recording results are dynamic and can get out of sync with the actual screen of the user. That leads to 37.4% of screens which contain invalid objects. This induces invisible or misaligned objects, or objects which are not clickable (greyed out). The researchers filtered invalid objects by training a Residual Neural Network (ResNet) model with the screenshots to classify nodes as invalid if their bounding boxes don't match. Also they introduced two models: a Graph Neural Network (GNN) and a Transformer model to each determine the view type (also related to the view class). For that they considered the view hierarchy attributes as well as the screenshots via a Convolutional Neural Network (CNN). They claim they outperform heuristic approaches for detecting layout objects without a visual valid counterpart and also can recognize their types in more than 85%. This pipeline could help to improve intent prediction algorithms as less inconsistent data is applied to the model.

3.2 Vector models

3.2.1 Screen2Vec

Toby Jia-Jun Li, Lindsay Popowski et al. [24] wrote a NN called Screen2Vec which embeds the UI components while preserving the semantics. It is claimed that they are among the first to develop a NN for mobile screens which takes textual, visual design, and layout patterns and app context meta-data into account. As inspiration they used the Word2Vec to predict result by considering the context and map them to a Continuous Bag-of-Words (CBOW). The self-supervised model consists of two pipeline levels. The outer level (Graphical User Interface (GUI) screen level) combines embeddings of GUI components, layout hierarchy and app descriptions. The inner level is only present for the GUI components as they contain nested embeddings for the screen text and the class type. The screen text (in the inner level) as well as the app description (in the outer level) is processed using a pretrained Sentence-BERT model. The layout hierarchy is converted to a colored image

Key	Type	Shape	Description
			Per View
activity_name	string	(1)	Name of the activity: e.g. “com.my_app.AppName.MainActivity”
is_keyboard_deployed	bool	(1)	Indicates if the keyboard is shown
request_id	int	(1)	Id used by the crawler to request the view
			Per Node
abs-pos	bool	(1)	Indicates if position in *bounds* is relative or absolute; if *true*, *rel-bounds* is set
adapter-view	bool	(1)	Indicates that children are loaded via an adapter, see [7]
ancestors	[string]	(None)	Ancestors of current node, e.g. “android.view.View”
bounds	[integer]	(4)	Absolute or relative boundaries, dependent on *abs-pos*
children	[node]	(None)	Child nodes
class	string	(1)	“com.my_app.lib.ui.views.DropDownSpinner”
clickable	bool	(1)	User can interact by press / click
content-desc	string	(1)	(Accessibility) description of the node “Interstitial close button”
draw	bool	(1)	Indicates if this node is drawn on the canvas
enabled	bool	(1)	Indicates if this node is in the enabled state
focusable	bool	(1)	Indicates if this node can be focused
focused	bool	(1)	Indicates if this node can is currently in focus
font-family	string	(1)	States the font family, e.g. “sans-serif”
long-clickable	bool	(1)	Indicates if this node has a long press action
package	string	(1)	States which packages the node belongs to “com.my_app.mypackage”
pressed	bool	(1)	Indicates if this node can is currently pressed
rel-bounds	[integer]	(4)	Relative boundaries, if *abs-pos* is set to *true*
resource-id	string	(1)	The unique resource identifier for this view “android:id/navigationBarBackground”
scrollable-horizontal	bool	(1)	Indicates if this node can be scrolled horizontally
scrollable-vertical	bool	(1)	Indicates if this node can be scrolled vertically
selected	bool	(1)	Indicates if this node can is currently selected
text	string	(1)	Text value if this node is a textual element
text-hint	bool	(1)	Explanation text for text boxes or icons
visibility	string	(1)	Indicates if this node is hidden, e.g. “visible”, “gone”
visible-to-user	bool	(1)	Indicates if this node can be seen in the viewport by the user

Table 3.1: Collection of attributes of a *view hierarchy* record, extracted from all interaction traces of the Rico [5] dataset.

encoding the text and image boundaries with colors (like in 3.1.2). With such model, a vector can be calculated for each screen which then can be compared to each other, e.g. by the euclidean distance between the pixel representations, or comparing the distance in the view hierarchy representation. When taking all features into account, both, the euclidean and the hierarchical approach, get an accuracy of around 0.85 that the correct screen is among the first 1% of the models predictions. In around half of the cases the predicted screen ("Top-1 Accuracy") is the correct one.

Such an approach of representing an Android layout and context in a vector can be used as pretrained embedding for feeding a RNN predicting upcoming screens.

3.2.2 Screen2Words

Similar to [24] Screen2words [33] considers the screen context from the view hierarchy to create a screen embedding. The goal is to provide a summarization for an unseen screen by usage of it's app and UI context. They describe such a technique as multi-modal, as it "leverages input from multiple data sources", like screenshot images, textual labels and UI tree structures. As basis for their training data the Rico-SCA (section 3.1.2) dataset is used, to remove inaccurate view hierarchies. Although the public code did not provide a direct way to parse the Protocol buffers of Rico-SCA. In addition they hired 85 labelers to manually annotate each of the 22k existing screens with multiple summarizations, resulting in more than 100k phrases.

As input for their summarizing NN they used a flattened view hierarchy which contained padded embeddings for categorical values, like the view class. Also the textual components of the screen were extracted and encoded with a pre-trained GloVE Word embedding. The GloVE encoding was also applied to the app description. The description was combined with the textual components and the other screen attributes to serve as input of a Transformer Encoder . Simultaneously each screenshot was embedded though a CNN, which was then concatenated with the Transformer Encoder to form the overall encoder of the Screen2words model. The screen summaries were used as labels for the model and were also embedded with the GloVE encoding.

As human validation they made a study consisting of more than a thousand participants. These had to give a star rating from one to five to assess the quality of the screen summarization. They also trained different graduations of their model by removing some of their modalities. That showed that using all modalities brought the best results (3.4 stars mean) in their rating. Using only the screenshots (pixel) modality the ratings were lower by almost one star indicating that adding more modalities contribute to improve the screen to word vectorization.

3.3 Time Series / Sequence models

3.3.1 Click Sequence Prediction / PathFinder

Seokjun Lee et al. [21] propose a technique called *PathFinder* which aims to predict the sequence of user clicks in Android mobile apps. The user input and the contextual data is collected via the Android Accessibility Services (2.1.2), so the users OS does not need to be modified or rooted. They collected the data from 55 students of their university with a sequence tracing tool and collected near 2 million button clicks from over a thousand apps. They follow a collaborative and content-based

approach which takes both all the users data as well as the individual preferences into account. The *button depth* describes the number of clicks or taps until the user gets to their target screen. In average nearly the user has 16 buttons as candidates to press as the next action. With a personalized UI a the *button depth* should decrease significantly. The next user click is dependent on very recent but also on previous clicks happened a longer time ago, e.g. taking a picture relates to uploading it later to their Social Networking Service (SNS). The authors train a LSTM model to predict the next button, which will be clicked on. PathFinder predicts the most probable three buttons with a 0.76 F-measure.

In contrast to this work, *PathFinder* does not take into account the complete view hierarchy or other spatiotemporal information. Just the previous and the current app and the click history with their button properties are considered. Also as far as known the dataset and code is not publicly accessible.

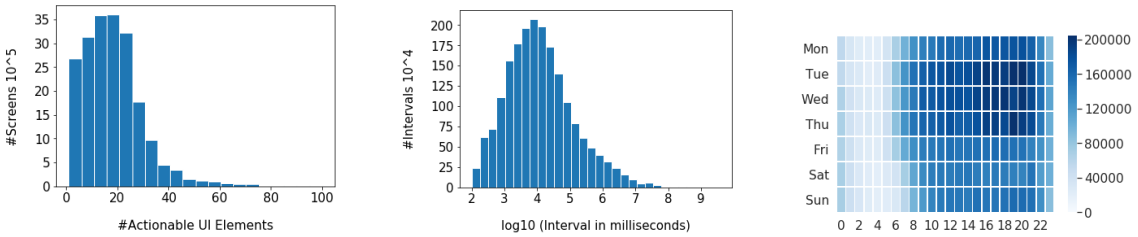
3.3.2 Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning

Xin Zhou and Yang Li [34] extend the work of [21]. They expect to optimize the UI experience by recommending the users next click interaction based on their findings. They gathered a dataset of 20 million clicks from 4k mobile users. The goal was to overcome the challenge of accurate but also scalable click sequence modeling. That means that apps are not limited in their composition and the screens get increasingly diverse and there's no predefined set of UI elements. Also the users click behavior is very individual and heavily dependents on situational factors.

Based on the Transformer architecture in they created a deep learning model which has 48% accuracy for predicting the next user click and 71% accuracy for the three most probable *actionable* objects. The researchers differentiated three main inputs for embedding their elements visible on the screen: the text content, the type of view and the bounding box. All the elements are then passed to a Transformer Encoder representing a single screen. Together with the click event as well as the time encoding the screen embedding serves as one concatenated input step. The encoded time can be very different as it doesn't follow a regular sample interval, but is recorded as soon as the screen is changing 3.1b. Multiple input steps then form a sequence fed in a second Transformer Encoder which contains all past screens and clicks. The current screen embedding and time are then passed to a pointer (M-layer perceptron) which calculates the most probable *actionable* elements to be clicked on. They consider a UI element as *actionable*, if it is currently `clickable`, `visible` and `enabled` (cf. table 3.1). More complex screens can have much more actionable elements, which makes the prediction much more difficult (figure 3.1a).

This paper solves a lot of problems previous attempts had. The dataset includes cross-app transitions which make 26% of all clicks, which are also considered in their model. The current context was taken into account such as the time of the day and day in the week 3.1c which adds a lot of semantics. Further they used a transformer model with self-attention, which reduces the training times significantly compared to LSTM. Compared to the approach tested in this paper, they also predict the concrete element that will be clicked on instead of the absolute screen coordinates.

Unfortunately as far as the investigation permits, neither the dataset nor the code were provided publicly.



(a) Number of actionable elements per screen

(b) The distribution of time intervals between click events

(c) The distribution of time intervals between click events

Figure 3.1: View element insights [34]

4 Methodology

In the upcoming section the methodology for this thesis is presented. This includes the type of work, the analysis process of related work, how the data was selected, what decisions were made to accomplish the research goals, and how the research questions are answered.

Research Problem

Working with mobile OS and how the user can be supported by solving their tasks on digital end devices is the main booster to embrace to this topic. This also raises the question, how much aid a user can be given, to what detail it is capable of and where are its limitations. To this date and as far as known no open accessible prediction model was published to predict the user intent to the detail of their gesture inputs. Also, it is not apparent which technologies and methods are suitable to develop such an intent prediction system. In order to converge solving these problems, the research questions were developed.

Type of Work

The thesis follows a mixed strategy to develop the research topic. Qualitative statements were taken into account in order to explain existing approaches and describe theories. But the experimental proof of concept was evaluated using quantitative metrics. A study to evaluate the experimental findings was exceeding the scope of this master thesis, although it would be very interesting.

Research Approach

In order to have an overview of the subject, various research papers were worked through. These then were classified by its relevance to the topic and – if suitable – presented in the related work (chapter 3). Some of these papers included sample code and datasets, which were evaluated in parts, such as Rico (3.1.2), Screen2vec (3.2.1) or Screen2words (3.2.2). Also, tools were looked for, which can help accomplish the research. To acquire the basics of ML, a workshop has been passed, which teaches the handling with Tensorflow and Keras using the programming language Python. The structure of the thesis was chosen by knowledge of working on previous thesis, the given template, and proposals from other scientists. A proof of concept has been developed in order to provide a working example for intent prediction. This was evaluated with well-established metrics, such as the RMSE. The importance of the result was discussed and the research questions were answered.

Data Aquisition

Several factors affected the decision for the dataset, that is used to train the ML model, which are explained in the following.

A prototype app has been developed which could inspect the accessibility tree, and therefore a potential candidate to create a self-made dataset. Unfortunately, gathering data on its own could not be realized, as of a few reasons. Firstly, the group of expected participants would be too small

to fulfill the requirements for the model. Secondly, the infrastructure has to be built up and the app service has to be maintained, which is not manageable in the given time frame. Last, legal and technological barriers may raise during realization, like privacy regulations or overstressed servers.

The dataset of ERICA (3.1.1) is more suited, as it is quite extensive and contains human interaction traces. The successor dataset, Rico (3.1.2), is even bigger, which matches the requirements for ML. Unfortunately both, ERICA and Rico, don't provide cross-app interaction traces, restricting it to possible intent predictions only during using an app. They also weren't reflecting a real world application as they used a web-service approach, rather than the android accessibility services to gather their data. Because of the lack of large appropriate datasets the Rico dataset was chosen for training the model.

Technologies

The abstraction interface of Tensorflow, Keras, was chosen to implement the proof of concept. The popularity and public knowledge, but also the usage in other related projects, convinced to use Keras, which is built up on Python. Regrettably Python is not *type safe*, which made debugging unnecessarily hard and raised major problems during development of the proof of concept. For example, the input shape and the types of embedding layers is not inherent in the parameters. Therefore, the required structure of the input data for various layers cannot be determined beforehand, unless having major expertise, which results in unpredictable states or errors. As Integrated Development Environment (IDE) IntelliJ was used in combination with Jupyter Notebook platform. First for calculations of NN a Windows-laptop was used, which has limitations to run Tensorflow with a Graphics Processing Unit (GPU) and CUDA [31]. Then a Linux-machine with an Intel Core i5-6600 processor and a NVIDIA GeForce GTX 970 graphics card were used.

5 Results

This chapter will explain how a prediction model for user intent can be designed and what factors of influence have to be considered. The design will be shown on the basis of a proof of concept implementation.

As noted in the introduction the term *intent* has a very wide scope. It's prediction can only be made in fractions, or serve as indicator. The semantically closest way and also the most detailed would be to describe the users intent in words, thus a description of what the user wants to do next. Unfortunately, the users intent description cannot be determined yet, as no according dataset is provided. An important step, the screen summarization as shown in Screen2words 3.2.2 [33], already was made, which could also be fed with the users intention descriptions. But this would then only reflect the already passed fulfilled intent and not the upcoming next purpose. Also, the users flows can be predicted as shown in the works of ERICA 3.1.1. Next app prediction already works quite well [19], which also reflects the larger intent of the user, but is not very detailed. In the process of the working on this topic, surprisingly a successful attempt was made to predict the next user interaction, which is explained in section 3.3.2 [34]. This shows, that it is possible to predict certain user actions upfront.

In order to approximate to reflect the user intent the assumption is made that the next **user interaction** also give hints on what the user is intending to achieve. The interaction is resulting in another screen, which then the user (hopefully) intends to see.

To provide a basis for further development on this topic, the following criteria were determined.

- Make the model as independent of the data as possible (generalization).
- Make it extensible for solving other problems, e.g. run it as form a classification or apply reinforcement.
- Make it accessible to other developers.
- Make it reproducible by providing the correct sources and add documentation.

5.1 Datasets

The most relevant datasets already were presented in section 3.1. During research a few problems arose, which made it difficult to obtain a dataset which serves the needs of intent prediction. Most suited datasets, referenced in the papers (by Google and Samsung), were not publicly available. A promising candidate is the Rico dataset (section 3.1.2), as it provides a huge amount of traces, which is needed for successfully learning a NN. It is also quite up-to-date, which reflects current development and trends. Trade-offs had to be made in the correctness of the data. Some samples are missing or don't match with the screenshot.

To overcome these limitations, it is proposed to create a custom dataset, which takes advantage of the accessibility services (section 2.1.2). Traces then can be recorded during transitions of apps (cross-app) or while interacting with the OS. The length of a trace would then be extended to a session, defined from activating the screen until it is turned off (by the user or the system itself). Tracing with accessibility services also has the benefit, that the model could be used locally on the device without communicating with an external provider. Additional communication and storage of data on servers increases the risk of attacks and leakage, which should be avoided by any circumstances. On-device processing also is possibly faster, works without internet connection, and makes the application more trustworthy. A local working technology also facilitate reinforcement learning (section 2.2.2), which improves the user experience significantly. Additional features then could also be considered, like sensor data, such as Global Positioning System (GPS) and gyroscope, acceleration, temperature, light and many more. A user working with web applications or playing directly rendered games still is a challenge to track, as the accessibility service does not have access to these elements in the same way as for native apps.

5.2 Preprocessing Android UI Tree Data

The widely used dataset Rico was chosen as basis to feed the model. As noted in section 3.1.2, the structure is disaggregated in *app*s, containing *traces*, containing *view hierarchies*, containing *view*s. A significant aspect is, that the samples in the traces don't get mixed up with other samples, to keep the temporal dependence. The dataset already was analyzed and manipulated by other researchers, though some of the steps were adopted from Screen2words (section 3.2.2). This implicates the feature extraction from the nodes, the bounding box normalization, and the node flattening process (cf. section 2.2.1). The attributes of the Rico dataset (table 3.1) not exactly matching the accessibility tree (cf. listing 2.1), but they are similar enough to state that the process is applicable for both variants.

For the proof of concept these attributes were extracted: `bounds` (inherits four values), `visibility`, `clickable` and the `class`. Also, the nodes depth in the dom tree was calculated, apparent as `obj_dom_pos` and the visibility for the user was validated with `visibility_to_user_seq`. One screen exists of multiple views / components, which all have their own attributes. To be able to examine each screen as one sample, the view hierarchy had to be flattened. This was done by separating the views by attributes and merge each value by its feature. Therefore, a screen sample was converted to a list of features, which again holds a list of values. This is an important step to realize as the dataset now consists of four array dimensions, which plays a big role in later steps, as it makes further processing more challenging. The gesture traces were acquired from the dataset to provide them as labels, but also as additional input. They contain one (click) or multiple (swipe) coordinates. For simplification only the first value was considered, but it is proposed to also provide the other values as input. Then all traces were iterated through and samples were filtered out, which did not provide any features. Also, only traces were picked, which did provide six or more screen samples, to be able to train a sequential model, resulting in 4278 traces with 50301 samples. Then the features consisting of arrays were padded to the same length (here: 500). This could be done per feature, but it was uniquely applied, because the number of views and therefore the length for each nested feature array is the same.

The data then was split in a train and a validation set. Due to the limiting capacity of the GPU, only 20% of the traces were used. Therefore, 642 traces were selected for training and 213 for validation. The following processing applied equally, but executed separately for train and test split.

The input and output (label) pairs were generated, which had to be done after splitting the traces to keep the temporal order. Therefore, for each trace the first samples s (here: 5) acted as the input sequence I and the next one (here: sixth) was the label L (cf. figure 5.1). This step was repeated by shifting the samples by one, so that the first sample was dropped and the label from the previous record then was part of the second input sequence. As label then served the next (here: seventh) sample. This process was done until no sample was left in the trace, that could take the role of the label. For the case of the proof-of-concept all features were dropped from the labels, except the click / gesture position values f_1 and f_2, because that's the prediction that has been looked for. After the input and output pairs for each trace have been generated, they then could be finally merged together in one array. The reason is that the input sequences and labels were then fixed and didn't depend on the upcoming samples in the trace.

5.3 Model

In the proof-of-concept a very dynamic approach was selected to be able to support new features without major changes. Therefore, depending on the input features f, the model was adapted automatically by these rules (cf. figure 5.1):

- If the sample feature consisted of a single value, it was directly passed for further processing.
- If the sample feature included multiple values, they were reduced in the dimensionality (dense layer) based on the length and provided each output value as a separate feature dimension.
- If the sample feature provided a continuous numerical value (normalized), it was also directly passed to be processed further.
- If the sample feature provided a categorical value (indexed), it was embedded (section 2.3.1) and then flattened to match the array dimensionality.

For simplicity all categorical values were handed over as array as it would not make a difference in processing. The numerical and categorical embeddings then were concatenated together and handed over to the LSTM. The LSTM expects an array / space dimension of three. The first array dimension is the sample dimension (unlimited), which in this case is the list of all sample sequences. The second array dimension is the time step dimension (fixed size), thus the sample sequence itself. The third array dimension is the feature dimension (fixed size), therefore there's no dimension left to encode nested or even variable arrays, such as multiple views / components. For that reason multiple embedded category values had to be flattened (reshaped) before entering the LSTM model with 128 neurons.

The output of the LSTM then was passed to a dense layer with 32 neurons, followed by a dropout of 20% of neurons. Finally, the values were passed to the output layer with two neurons, which were representing the x (Gest_x) and y (Gest_y) axis of the predicted gesture.

Due to all the numerical and categorical features chosen from the dataset, the model **SelectedFeatures** in figure 5.1 crystallized.

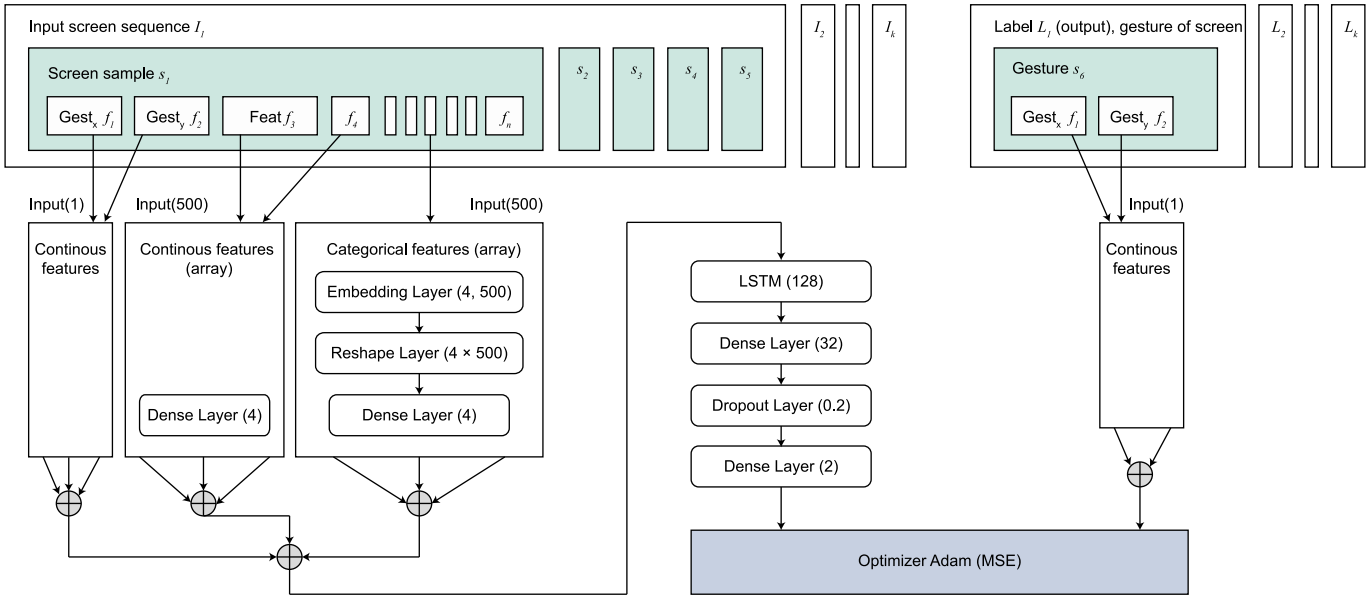


Figure 5.1: Structure of the proposed intent prediction model, I_k is the kth input sequence, L_k is the kth label from the sequence, s_m is the mth sample in the sequence, f_n is the nth feature of a sample.

The above structure also allows to take in more features as prediction. Then it has to be reconsidered, at to which screen the gesture or click belongs to. Is it wanted to predict the next screen from the current screen and its gestures? Or is it desired to predict the gestures made by the user on the current screen and also predict the views of the next screen?

While the design process of the model, multiple approaches were considered.

One procedure would be to use a pretrained embedding for Android UIs, e.g. the Screen2vec embedding of [24] in section 3.2.1. The big advantage is, that the screens are already reduced to a vector and the RNN, like the LSTM, has much more valuable inputs and therefore requires less time for training. This also comes with a disadvantage: the backpropagation process cannot be applied to the pretrained model and the features which are important for training a temporal dependent model may differ from the needed features e.g. similarity of screens based on an Autoencoder (AE).

More approaches have to be considered, when not just want to predict the click position, but rather a view element or even the whole screen should be forecasted. Then due to the high dimensionality of the output, it could make sense to precede an AE net before the RNN layer, to work out important features. This would have a similar structure as the pretrained embedding, but is adaptable during training. The decoder of the AE then also can be moved to after the RNN layer, which should improve the performance. Or the encoder and the decoder are subsequently processed afterwards to provide all raw feature inputs to the RNN.

As earlier mentioned, an even more promising approach, which has been proved to work well in click sequence prediction (section 3.3.2), is the transformer.

5.4 Evaluation

To get an insight of what features are providing valuable information, a second model was built, which only included the click / gesture sequence as feature, referenced as **ClickOnly**. During preparing the dataset an important step was missing, which could help improving performing better. Only the click of the next screen was predicted, but not the click on the current screen. Therefore, all click values were shifted by one step and filled up with a value of 0.5 which represents the mid of the screen. The idea is that the click now belongs to the screen which is shown after the action. In theory that should improve the models performance, as it then can choose the click based on the elements shown on the current screen while predicting. This experiment is referenced as **FeaturesClickShifted**.

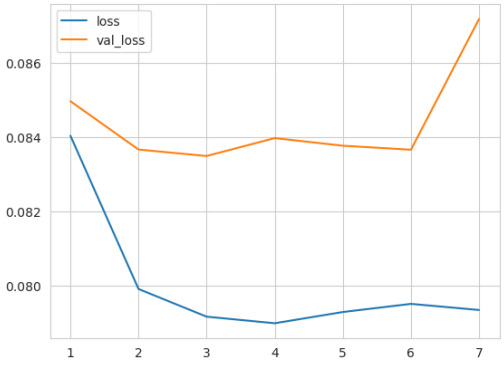
In the figures of 5.2 the training loss and the validation loss can be seen for all three approaches. A numbered representation of the last epoch for each model is listed in the table 5.1. The models all were trained with the Adaptive Moment Estimation (Adam) optimizer by using the MSE as loss metric.

In all three models, the loss was around 0.09 for the validation set. This is a deviation of 0.3 for the RMSE. That means for a screen from left **0** to right **1** and bottom **0** to top **1**, the predicted point (click) was around 0.3 away projected to the euclidean distance. To put it in perspective, if the center is used as prediction, it is off by a maximum of 0.5, if the expected user click is always at the border. But in the mean the clicks are even closer. To check that the RMSE was calculated, if all predictions were made to 0.5 for Gest_x and Gest_y, resulting in 0.3102. Using the mean for Gest_x (0.4360) and Gest_y (0.4552) resulted in a RMSE of 0.3052. This implies that the models predicting the user clicks only slightly better than the statistical mean. In contrast to the expectation, shifting the click sequence in FeaturesClickShifted did not result in much better predictions compared to the SelectedFeatures model, at least not with the small amount of training data. Even the ClickOnly model performed better, with no additional features. As buttons are scattered all around the screen, 0.3 RMSE seems a reasonable value, if the models always predicts the mean as the next click interaction.

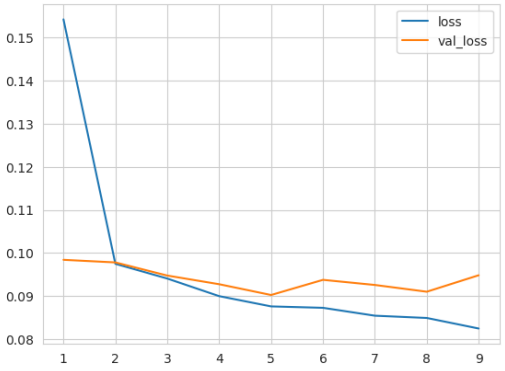
Regarding the number of epochs, the ClickOnly model needed less than the others, which is expected as it has only two feature dimensions. The other two models needed more epochs to align with the values of the ClickOnly model. This can be the result of aligning the additional random weights for all the additional features. The FeaturesClickShifted model needed slightly more epochs than the SelectedFeatures model, which could be by chance, or it found slightly more correlations than the SelectedFeatures model.

Method	Loss	Validation Loss	RMSE Val Loss	Number of epochs
ClickOnly	0.0793	0.0872	0.2953	7
SelectedFeatures	0.0824	0.0948	0.3079	9
FeaturesClickShifted	0.0811	0.0900	0.3000	10
Center	–	0.0962	0.3102	–
Mean	–	0.0931	0.3052	–

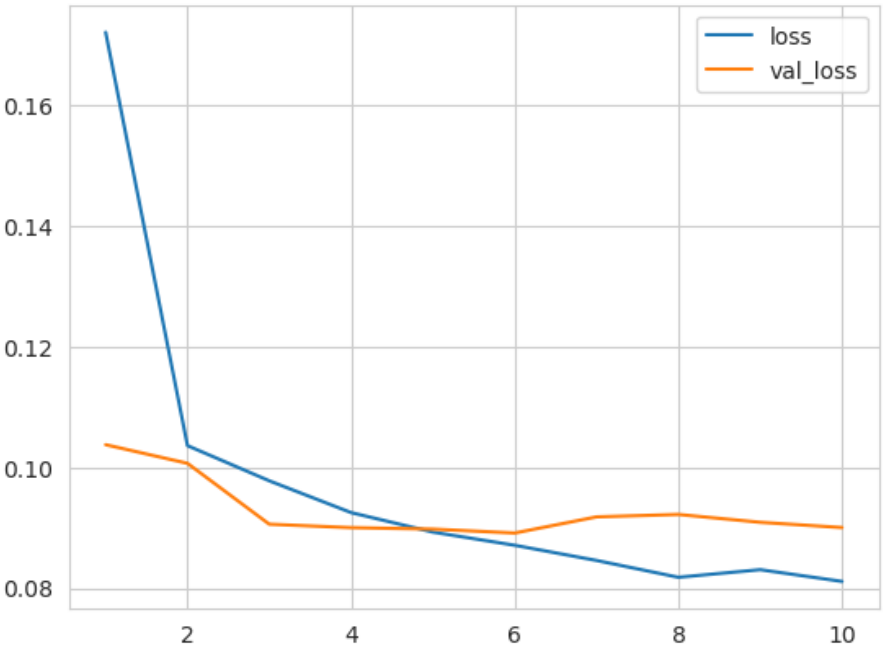
Table 5.1: Training and validation loss (MSE) of all three models, ClickOnly, SelectedFeatures and FeaturesClickShifted



(a) ClickOnly: Only clicks as features



(b) SelectedFeatures: Bounding box and click features



(c) FeaturesClickShifted: Bounding box and shifted click features

Figure 5.2: Training loss (*loss*, blue) vs. validation loss (*val_loss*, orange). Horizontal axis is the number of epochs. Vertical axis is the loss as MSE.

6 Discussion

In this chapter a subsumption is presented which puts the results of the proof-of-concept into perspective.

The most limiting factors are the dataset, the computational performance, as well as the development time. As mentioned previously the underlying dataset plays a significant role, how the model performs. Because the dataset does only consider app traces, predictions while using the OS itself cannot be made. Also, some view changes while using an app were not recorded, resulting in a gap between training data and the real world application. Rico didn't provide data about paid apps or apps with the need for an account, resulting in not reflecting the day-to-day scenarios. Nonetheless, this was not the bottleneck of the research.

While evaluating the three models ClickOnly, SelectedFeatures and FeaturesClickShifted, all of them had a similar performance. Compared with the calculated statistical mean value of the click positions with the RMSE, the models only performed slightly better. This can have multiple reasons. At first, the limited GPU capacities allowed only to use 20% of the data, resulting in much less training and evaluation data. Second, the distance metric may is not really applicable to train and predict the next user input. This denotes that buttons are positioned very differently and a wrong prediction can be positioned far away. A much better way would be the conversion of model to a classification of actionable elements in order to predict the most likely component, which the user will press. This is also proposed in [34], together with a relative ranking metric, which is also more appropriate to calculate the error for predicting the next user action.

To improve preprocessing, more approaches have to be tested against, to work out the core parameters to predict the next user intent. Thus, the model needs more investigation on what kind of data is needed and what layers and methods are suited best. E.g. the number of neurons for embeddings is interesting, as well as adding additional features such as the screenshot through a convolutional embedding or adding pretrained word embeddings like done in Screen2words [33].

6.1 Suitable Models for Prediction of User Intent

To answer the question of *what a suitable model for the prediciton of user intent* is, LSTM has to be compared against other methods of predicting user intents. Classic stochastic approaches may can predict larger scope of the user such as the next app or a general user workflow. But they are not sufficient for predicting views, screens, or even precise gesture inputs. To overcome this limitation the ML-models have established in large datasets. As the described problem is to be contextualized in the prediction of time series, the following options are offered:

- Simple RNN
- Gated Recurrent Unit (GRU)

- LSTM
- transformer

Simple RNN is limited in the capacity of establishing long-term semantics. GRU is missing the forget gate compared to the LSTM making it simpler and faster, but may perform weaker on complex datasets. The transformer model has many advantages, such as fast and efficient training, parallelism of input sequences and recognizing long-term patterns through multi-head attention. On the other hand few prior work was done in the mobile sector which covers prediction of UI trees. Therefore, no publicly available coding approach was found which could be extended. Also, as described in [34] the structure of the model is quite complex – with two transformers – and needs more processing steps in general. LSTM is well documented and the common choice to predict time dependent series. It is well-supported by Keras and easy to use. Also, similar approaches have been made in the area of app prediction or app summarizatio, which can be used as basis for this work, such as Screen2Words 3.2.2.

6.2 Level of Detail of Intent Predictions

The matter to what level of detail user intent predictions can be made, was already discussed in parts in the results chapter 5 to work out the missing development for intent prediction.

As mentioned, only hints or parts of the semantic intent can be predicted. In the numerous papers, the app, the user flow and even the click sequence, which can be seen equatable to the user interaction, was proven to be predictable to a certain degree. Still missing is a prediction of screens, views and intent descriptions, which seems doable by combining the Screen2words (section 3.2.2) model with the transformer model (section 3.3.2). So not only the interaction, but also the next contents, such as screen or single views can give a hint to the intent. This is also possible, if replacing the labels in the proposed model, but it is not evaluated how precise these predictions will be.

7 Conclusion and Future Work

Summary

In this thesis the Android UI structure and the ways of retrieving them has been worked out, to be able to gather a meaningful dataset. The basics of ML, NNs, the preprocessing steps and evaluation metrics have been illustrated to grasp the idea of the proof-of-concept. The word intent has much room for interpretation, thus a set of indicators were given to better differentiate. A proof-of-concept for an intent prediction model based on a LSTM was worked out, which does not yet predict advantageous gestures, but only predicts slightly better mean click coordinates. The environment did not enable the performance to fully test out the potential. However, the proposed concept provides flexibility and extensibility for future work and is accessible for the public. More promising approaches arose, like transformer multi-attention model, actionable element prediction, and the usage of metrics like relative rankings (3.3.2). The available technologies already enable large parts of intent prediction. Nonetheless, more precise or semantically meaningful prediction models, such as screen or descriptive ones are still missing.

Outlook

Although the results of the proposed model did not quite match the expectations, this field of application has lots of open questions to be researched on. The prospects are by adding more semantics, such as language features or sensory inputs, any scope of intent can be predicted more granularly. The data selection also has lots of potential to improve by applying preprocessing steps such it is done in RicoSCA [25] or Clay [22]. Also the types of elements (class) have a much more impact on the outcome than the gesture positions. A public accessible app tracing app and a big enough dataset which also cover cross-app traces would be a good basis for future research. Other model types have to be evaluated or reproduced. Also, a user study should be conducted to evaluate how helpful an overlay can be, which proposes the next item to click. Today, smartphones are expected to be powerful enough and are equipped with intelligent processors to take tasks like recording the accessibility tree and user interaction prediction. This would also help preserving privacy. Zhou and Li [34] already provided a “Next Click Overlay” that reduces the number of traversals from 9.04 to 2.61. Feedback and improvement on such a visualization of a prediction system would be very interesting. Applying reinforcement learning (section 2.2.2) to the existing models also would help improving the user experience.

In regard to the current development in large language models like Bart or ChatGPT, it will be exiting to see what surprising breakthrough comes up next.

References

[1] Contributors of 'App ins Grüne'. *App Ins Grüne*. 2020. URL: https://github.com/mimuc/app-ins-gruene (visited on 01/01/2024).

[2] Saqib Alam, Nianmin Yao. "The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis". In: *Computational and Mathematical Organization Theory* 25 (2019), pp. 319–335.

[3] Flutter Authors. *Semantics class*. 2023. URL: https://api.flutter.dev/flutter/widgets/Semantics-class.html (visited on 01/01/2024).

[4] Jason Brownlee. *How to Use Word Embedding Layers for Deep Learning with Keras*. 2021. URL: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ (visited on 01/04/2024).

[5] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, Ranjitha Kumar. "Rico: A mobile app dataset for building data-driven design applications". In: *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 2017, pp. 845–854.

[6] Biplab Deka, Zifeng Huang, Ranjitha Kumar. "ERICA: Interaction mining mobile apps". In: *Proceedings of the 29th annual symposium on user interface software and technology*. 2016, pp. 767–776.

[7] Android Developers. *AdapterView*. 2023. URL: https://developer.android.com/reference/android/widget/AdapterView (visited on 01/01/2024).

[8] Android Developers. *How Android draws Views*. 2023. URL: https://developer.android.com/guide/topics/ui/how-android-draws (visited on 01/01/2024).

[9] Android Developers. *Semantics in Compose*. 2023. URL: https://developer.android.com/jetpack/compose/semantics (visited on 01/01/2024).

[10] Android Developers. *Thinking in Compose*. 2023. URL: https://developer.android.com/jetpack/compose/mental-model (visited on 01/01/2024).

[11] Android Developers. *UI layer*. 2023. URL: https://developer.android.com/topic/architecture/ui-layer (visited on 01/01/2024).

[12] React Native developers. *Accessibility*. 2023. URL: https://reactnative.dev/docs/accessibility (visited on 01/01/2024).

[13] Dictionary.com. *Intent*. 2023. URL: https://www.dictionary.com/browse/intent (visited on 01/06/2024).

[14] Bao Tram Duong. *Fine-Tuning Inputs: Data Preprocessing Techniques for Neural Networks*. 2023. URL: https://baotramduong.medium.com/data-preprocessing-for-neural-network-0b398b43d309 (visited on 12/30/2023).

[15] Zoubin Ghahramani. "Unsupervised learning". In: *Summer school on machine learning*. Springer, 2003, pp. 72–112.

[16] Alireza Ghods, Diane J Cook. "Activity2vec: Learning adl embeddings from sensor data with a sequence-to-sequence model". In: *arXiv preprint arXiv:1907.05597* (2019).

[17] Rob J. Hyndman, Anne B. Koehler. "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22.4 (2006), pp. 679–688. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0169207006000239.

[18] Kasthuri Jayarajah, Youngki Lee, Archan Misra, Rajesh Krishna Balan. "Need accurate user behaviour? pay attention to groups!" In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 2015, pp. 855–866.

[19] Katerina Katsarou, Geunhye Yu, Felix Beierle. "WhatsNextApp: LSTM-based next-app prediction with app usage sequences". In: *IEEE Access* 10 (2022), pp. 18233–18247.

[20] Christoph Kofler, Martha Larson, Alan Hanjalic. "User intent in multimedia search: a survey of the state of the art and future challenges". In: *ACM Computing Surveys (CSUR)* 49.2 (2016), pp. 1–37.

[21] Seokjun Lee, Rhan Ha, Hojung Cha. "Click Sequence Prediction in Android Mobile Applications". In: *IEEE Transactions on Human-Machine Systems* 49.3 (2019), pp. 278–289. DOI: 10.1109/THMS.2018.2868806.

[22] Gang Li, Gilles Baechler, Manuel Tragut, Yang Li. "Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://github.com/google-research/google-research/tree/master/clay. New Orleans, LA, USA: Association for Computing Machinery, May 2022, pp. 1–13. URL: https://doi.org/10.1145/3491102.3502042.

[23] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, Huan Liu. "Feature Selection: A Data Perspective". In: *ACM Comput. Surv.* 50.6 (Dec. 2017). ISSN: 0360-0300. DOI: 10.1145/3136625. URL: https://doi.org/10.1145/3136625.

[24] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, Brad A Myers. "Screen2vec: Semantic embedding of gui screens and gui components". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 2021, pp. 1–15.

[25] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, Jason Baldridge. "Mapping natural language instructions to mobile UI action sequences". In: *arXiv preprint arXiv:2005.03776* (2020).

[26] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[27] Georgios Nanos. *Deep Neural Networks: Padding*. June 2023. URL: https://www.baeldung.com/cs/deep-neural-networks-padding (visited on 01/01/2024).

[28] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015. URL: http://neuralnetworksanddeeplearning.com/.

[29] Android Open Source Project. *AccessibilityNodeInfoDumper*. 2012. URL: https://android.googlesource.com/platform/frameworks/testing/+/jb-dev/uiautomator/library/src/com/android/uiautomator/core/AccessibilityNodeInfoDumper.java (visited on 01/01/2024).

[30] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, Jianhua Zou. "Deepapp: a deep reinforcement learning framework for mobile application usage prediction". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 2019, pp. 153–165.

[31] TensorFlow Team. *Install TensorFlow with pip*. 2023. URL: https://www.tensorflow.org/install/pip#windows-native (visited on 01/06/2024).

[32] Jesper E Van Engelen, Holger H Hoos. “A survey on semi-supervised learning”. In: *Machine learning* 109.2 (2020), pp. 373–440.

[33] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, Yang Li. “Screen2words: Automatic mobile UI summarization with multimodal learning”. In: *The 34th Annual ACM Symposium on User Interface Software and Technology*. 2021, pp. 498–510.

[34] Xin Zhou, Yang Li. “Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning”. In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys ’21. Amsterdam, Netherlands: Association for Computing Machinery, 2021, pp. 473–483. ISBN: 9781450384582. DOI: 10.1145/3460231.3474264. URL: https://doi.org/10.1145/3460231.3474264.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.