IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 7, 2022, accepted January 29, 2022, date of publication February 10, 2022, date of current version February 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150874

WhatsNextApp: LSTM-Based Next-App Prediction With App Usage Sequences

KATERINA KATSAROU[1], GEUNHYE YU[1], AND FELIX BEIERLE[2], (Member, IEEE)
[1]Service-Centric Networking, Technische Universität Berlin, 10587 Berlin, Germany
[2]Institute of Clinical Epidemiology and Biometry, University of Würzburg, 97080 Würzburg, Germany

Corresponding author: Katerina Katsarou (a.katsarou@tu-berlin.de)

We acknowledge support by the German Research Foundation and the Open Access Publication Fund of TU Berlin.

ABSTRACT Next app prediction can help enhance user interface design, pre-loading of apps, and network optimizations. Prior work has explored this topic, utilizing multiple different approaches but challenges like the user cold-start problem, data sparsity, and privacy concerns related to contextual data like location histories, persist. The user cold-start problem occurs when a user has recently registered to the smartphone app system and there is not enough information about his/her preferences and his/her history of smartphone usage. In this work, we try to address the above issues. We introduce WhatsNextApp, an approach based on LSTM (Long Short-Term Memory) networks using sequences of app usage logs. Our approach is inspired by Word Embeddings and treats sequences of app usage logs as sequences of words. We collect a real-life data set consisting of 975 Android users with over 22 million app usage events. We build a generic (user-independent) WhatsNextApp model and the evaluation with our data set shows that it outperforms related studies for existing users where we achieve a recall@8 (recall for the top 8 apps) of 92%. For the user cold-start problem with the 500 most frequent apps, we achieve a recall@8 of 82.7%.

INDEX TERMS Human-centered computing, smartphone, machine learning algorithms, LSTM.

I. INTRODUCTION

Every day, smartphone users use a variety of apps for different purposes [1]. The number of apps in Google Play and the Apple App Store already exceeds 2.8 million apps and 2.2 million apps, respectively [2]. In several studies, researchers attempt to tackle the problem of predicting the next app usage, but they face the limitation of poor performance of their approach in the case of the cold-start problem. The *cold-start problem* describes states that make the prediction hard or less accurate because there is a lack of training data to perform a prediction, for instance, when a user has recently registered to the system [3]. Furthermore, many Spatio-temporal features must be extracted to improve the prediction performance, raising privacy concerns, as the user has to provide more data.

Several actors are interested in knowing which apps the users are going to use. These actors include software developers, mobile OS operators, and network operators. We see three main benefits from predicting app usage: user interface optimization for enhancing usability and accessibility, loading apps that are about to be used into memory or keeping them in memory, and optimizing network infrastructure, as different apps produce different amounts of traffic with different endpoints.

App usage prediction can help designers and software developers to improve user interfaces. The point is that smartphones can figure out the next app based on repetitive patterns. For instance, the smartphone can be aware that a calendar app will open because a user usually uses the app when the user gets a call or after checking an email app. Therefore, some smartphones are released embedded with functions of next app suggestions based on app usage in the user interface such as app suggestion by Siri on iPhone [4]. However, if the next app is predicted more accurately, users could get a more efficiently operational interface with the next app to be used.

Another applicable case for app usage prediction is related to pre-loading the next app into memory. By pre-loading those apps that are predicted to be used next, opening times can be decreased. For example, people frequently experience the fact that some heavy apps (e.g., gaming apps) take

The associate editor coordinating the review of this manuscript and approving it for publication was Li He.

over 20 seconds to launch and need more seconds to be used [5]. Moreover, not only overloaded apps but also even simple apps such as weather apps require at least 10 seconds [2]. As a result, if those circumstances depend on app prediction, the system can load the next apps in advance, and people can use their app without considerable waiting periods.

Also, app usage prediction is helpful for network resource allocation. On the business side, the communication carriers exploit app usage prediction for customizing bandwidth allocation schemes [2]. Their service quality relies on offering stable bandwidth. However, the bandwidth has changeability depended on the time and locations of smartphone usage [2], [6]. Hence, if they can predict the next app usage, carriers could proactively determine the bandwidth allocation. On the user side, through correctly cached network contents of the predicted apps, the app content could be provided with much higher network speeds [4].

In this work, we propose the WhatsNextApp approach, a deep learning approach based on Long Short-Term Memory Networks (LSTMs) that considers the sequences of app usage and is inspired by the concept of word embeddings. The approach is generic (user-independent) and addresses the problem as a multiclass classification problem. WhatsNextApp is applied to a dataset of 975 Android users with over 22 million app usage events and 19,485 distinct app package names, collected from October 2018 to June 2019. In particular, we take sequences of apps that occur within specific time windows of 5 minutes, 1 hour, 12 hours, one day, and one week and we find the mean app usage within these time windows for constructing sequences of fixed lengths. Our main contributions are:

- Our approach, WhatsNextApp, with a prediction performance of 92% for the top 8 apps outperforms the methods presented in the existing literature.
- We address the cold-start problem as our approach is generic (user-independent), and the next apps of new users can be predicted. Moreover, we achieved higher prediction results for the cold-start problem than the state-of-the-art methods, particularly a recall@8 of 82.7%.
- We predict the next app usage only by using temporal features without including more Spatio-temporal features like location and smartphone activity as many state-of-the-art approaches do.
- We investigate the case when we predict the right next app and several next apps (specifically the next ten apps) using an LSTM encoder-decoder.

The rest of this paper is structured as follows. In Section II, we present the related works and their limitations. Then, in Section III, we describe the WhatsNextApp approach in detail, and in Section IV, we describe our data set and gain meaningful insights of our data. Finally, in Section V, we present the results of our extensive experimental evaluation and in Section VI our conclusions and the further work that can follow.

II. RELATED WORK

Baeza *et al.* [7] develop a model that uses sequences of apps and Spatio-temporal features like time, latitude, and longitude. The authors use the Parallel Tree Augmented Naive Bayesian Network (PTAN) as a predictive model because it uses correlations among attributes. For the user cold-start problem, they apply the Most Similar User Strategy based on collaborative filtering. Their approach achieves approximately 90% precision for the next app prediction but only 45.7% precision for new users.

Prior works employ Markov models in their approaches [8]–[10]. The authors use app transitions or app sequences for their prediction models. A global Markov Model lacks capturing diverse transition patterns, whereas a fully personalized model suffers from the sparsity of the observed transitions. In this context, there are efforts to make a cluster-level Markov model by clustering users' one-step app transition and compute a representative Markov model per cluster [11]. Despite the advanced attempts, Markov models suffer from insufficient training data with different lengths of sequences when dynamic changes happen [1]. To overcome the challenges, Parate *et al.* [12] designed a system that not only predicts the next app but also prefetches it on the mobile phone. The authors apply text compression with Prediction by Partial Match (PPM), including a variable-length Markov-based predictor.

Zhao *et al.* propose AppUsage2Vec for predicting app usage [3]. AppUsage2Vec was influenced by the concept of Doc2Vec and predicts app usage with a multiclass classifier. The authors also highlight the cold-start problem that the personalized models cannot solve successfully. An app attention mechanism measures the contribution of different apps to the prediction of the target app, and a dual deep neural network uses the Hadamard product on user and app sequence vectors. A recall of 86% for the top 5 apps is achieved, with 8,739 users in the data set. The authors show that the generic model gives better results than the individual model when the sequence samples decrease, but they do not show the results in terms of recall@5 for the user cold-start problem.

Han *et al.* propose a recommender system called Predictor, based on incremental k-nearest neighbors (IKNN) algorithm for addressing the problem of decreasing accuracy due to the increasing amount of training data over time [13]. The dynamic conditional probability of the user similarity and the app periodicity is used to address the cold-start problem. For evaluating app usage prediction performance as a situation of incremental data, 89.3% of average accuracy is obtained with ICANN. For the cold-start problem, the accuracy is 53%.

Yu *et al.* focus on the app usage across various locations in an urban environment [14]. Publicly available points of interest (POIs) data is used to predict app usage at a given location. The authors employ a transfer learning model based on collaborative filtering. The approach gives 83% hit rate for the top 5 apps in each location. Fang *et al.* [15] use a topic model for converting app and users' preferences into latent vectors, and then the KNN algorithm is used. Furthermore,

TABLE 1. **Comparison of app prediction studies, top-K means the result is based on top-K selected candidate apps and "–" stands for unreported information.**

Ref.	Users	Apps	Periods	Algorithms	Models	Evaluations	Results	Cold-start
[7]	480	–	7M	PTAN (Naive Bayesian)	Personalized	Average precision	90.3%	App & user
[16]	420	1447	100D	LSTMs	Generic	Accuracy Top-(1-10)	(42 - 77)%	Generic
[15]	1083	588	3D	Chain-augmented Naive Bayes	Generic	Accuracy@Top-5	86.6%	–
[2]	1,788	1,633	1W	Graph-based embeddings	Personalized	Accuracy@5	84%	–
[13]	30,000	–	1Y	ICkNNM (k-nearest neighbours)	Personalized	Accuracy@5	89%	Personalized
[22]	38	–	1Y	Markov model	–	Hit-rate@3	60%	–
[20]	25376	–	91D	Similarity Network	–	Precision Recall	60% 60%	–
[17]	55	1198	4W	LSTM	Personalized	Coverage Accuracy F-score	95% 96% 76%	–
[18]	180	–	–	stacked LSTM	–	Accuracy@3 Accuracy@5	75.48% 86.43%	–
[11]	17,062	9,583	1Y	iConRank (Markov Model)	–	Recall@5	80%	Generic
[12]	34	–	–	APPM (Markov Model)	Personalized	Accuracy@5	81%	–
[19]	443	–	21D	Reinforcement Learning Neural Network	Generic	Precision Recall	70.6% 62.4%	–
[9]	52	–	49D	Markov Model	Personalized	Hit rate@8	80%	–
[14]	6M	10,000	7D	Transfer Learning	Generic	Hit rate@5	83%	Generic
[3]	8,739	2,000	3M	Appusage2Vec (Word2Vec)	Personalized Generic	Recall@5	83.5% 86.0%	Generic
[8]	80	59	–	Markov Model	–	Hit rate@5	65%	–

the prediction is built on a chain-augmented Naive Bayes model since the authors use sequential temporal data. Finally, Chen *et al.* [16] attempt to solve the user cold-start problem by using the app usage history from other users to find similarities between their latent vectors and the latent vectors of the new users. Lee *et al.* [17] use an LSTM-based framework for predicting the click sequences in Android Mobile Apps in order to improve user experience. Firstly, they use data from all users, and then they create a personalized model including the 100 most frequent apps. However, the paper does not investigate the framework's performance for usual problems like user cold-start problems. Lee *et al.* [18] utilize an LSTM-based framework for app usage prediction in dual display devices. The top 5-level accuracy is 86.43%, but they do not mention how they address the cold-start problem. Shen *et al.* [19] propose *DeepApp* based on reinforcement learning with decreased prediction time (by 6.58 times compared to the state of the art methods) and report a precision of 70.6% and recall of 62.4%, but they do not mention how they deal with the cold-start problem. Jiang *et al.* [20] use app usage similarity networks using data from 25,376 users, and they accomplish almost 60% precision and recall.

Some of the studies analyzed above utilize a small number of smartphone users, or they collect data in a short period [8], [9], [12], [13]. Furthermore, the user cold-start problem is an open issue, even though the proposed solutions improve it significantly [13], [19]. Additionally, models based on Bayesian Networks [7] cannot react flexibly to changes in the app usage over time. Also, models that use features like location should include publicly available Points of Interests (POIs) in order to make the model applicable [14], [21]. Lastly, almost all of the studies focus on predicting only the first next app.

Table 1 shows an overview of these approaches. From the papers presented in Table 1, six out of the sixteen reviewed papers (37.5%) are dealing with the user/app cold-start problem. Additionally, five out of sixteen (31.25%) propose a Markov-model-based approach, and three (18.75%) propose an LSTM-based model. The number of users ranges from 34 to 30,000 and the period from 7 days to one year. Some recent studies focus on LSTM-based approaches [2], [17], [18], probably due to the increased popularity of LSTMs with sequential data. Finally, seven out of sixteen papers (44%) present personalized models, whereas five out of sixteen (31%) generic models.

WhatsNextApp uses a deep-learning-based approach that exploits LSTM networks and considers temporal features and specifically the sequences of app usage logs. Furthermore, by not having data from the same user both in training and testing sets, we address user cold-start problems with results that outperform state-of-the-art approaches. For example, for predicting the next app, we achieve highly accurate results for existing users by using only the sequences of app usage without including more Spatio-temporal features like location and smartphone activity. We build a generic model that does not depend on the user. Also, we implement an encoder-decoder LSTM that predicts for several next steps the app usage, whereas in most of the related works, only the first next app is predicted [13], [14]. Finally, by using a deep learning approach, we avoid problems related to collaborative filtering such as data sparsity [11], [13].

III. WhatsNextApp: APP USAGE PREDICTION USING SEQUENCES OF APP USAGE

Smartphone users spend a lot of their time on their smartphones as the numerous apps can satisfy various everyday needs. However, these apps could be not functionally related to each other, such as gaming and camera apps, or they could work harmoniously for specific aims or activities such as shopping and bank apps. Consequently, app usage appears in time order, and an app is frequently used in combination with other apps to meet specific needs [3]. In this section, we present the WhatsNextApp approach. First, we deal with the problem of predicting the next app as a supervised multiclass classification problem. In this context, we use LSTM-based models. Next, we describe the LSTM models that we use in detail and explain why they are suitable for dealing with a sequence of apps, ordered by time. Then, we explain how we determine the fixed sequence length of apps and transform the app names to word embeddings using the embedding layer of the LSTM. The model predicts the next app based on these sequences but without considering the user related to them, as it is a generic model. Next, we present the recall@k metric for evaluating the performance of our models. Finally, we use an LSTM encoder-decoder to address the problem by forecasting the next apps within a specific timing window. Figure 1 describes all the steps of this approach.

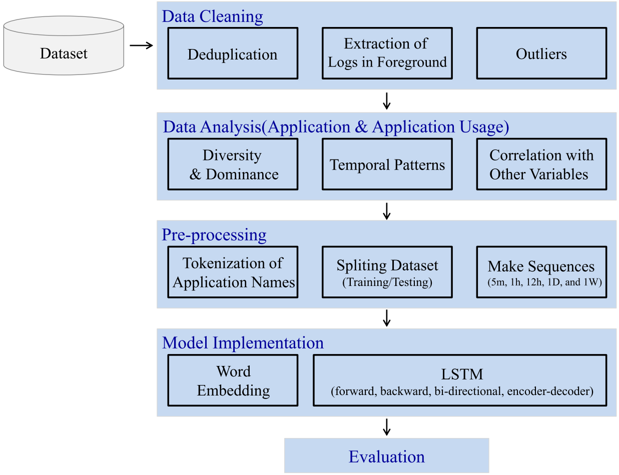

Dataset
Data Cleaning
Deduplication
Extraction of Logs in Foreground
Outliers
Data Analysis(Application & Application Usage)
Diversity & Dominance
Temporal Patterns
Correlation with Other Variables
Pre-processing
Tokenization of Application Names
Spliting Dataset (Training/Testing)
Make Sequences (5m, 1h, 12h, 1D, and 1W)
Model Implementation
Word Embedding
LSTM (forward, backward, bi-directional, encoder-decoder)
Evaluation

FIGURE 1. **Design overview of the model.**

Firstly, we conduct the data cleaning. The data cleaning step includes tasks like deduplication, extraction of helpful app logs (in foreground state), and treatment of outliers [23]. Then, the data analysis focuses on finding characterization and statistics of app usage with the cleaned data set. Hence, it is helpful to understand the principal streams and patterns of the data set.

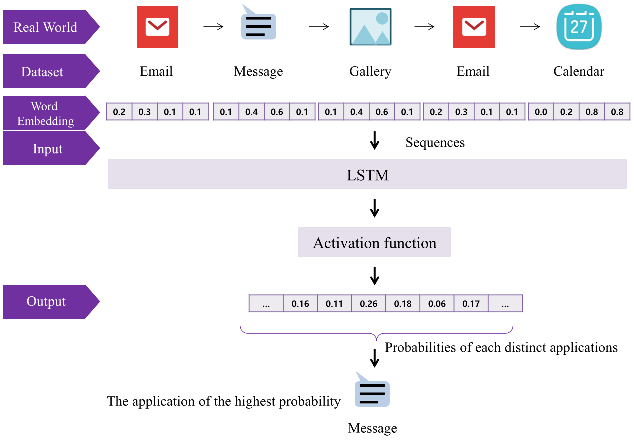


FIGURE 2. **Overview of WhatsNextApp.**

Figure 2 shows the structure of the WhatsNextApp approach, which is a generic model. In other words, every user does not have a personalized model. Instead, the generic model can be applied and predict the next app based on all users' data. The reason behind choosing to use a generic model is to solve the user's cold-start problem. The dataset contains 975 app usage logs. Furthermore, we split the data set into two parts, 90 percent of users for training and 10 percent for testing without any intersection. This splitting helps to correctly evaluate the model's quality and deal with the user cold-start problem. The model predictions are biased when data from the same users is used in training and testing sets. In the meantime, the model may poorly predict the next app of a new user. Given a set of apps A and an observed sequence consisting of n most recently used apps $(a_{r-n}, a_{r-n+1}, \ldots, a_{r-1})$, where each app a ϵ $\mathbf{A}$ predicts the next app a_r from $\mathbf{A}$. As a result, the conditional probability of the next app is:

$$P_r(\hat{a}_r | a_{r-n}, a_{r-n+1}, \ldots, a_{r-1}) \tag{1}$$

where a_{r-i} means the $(r - i)$-th app, ordered from the latest to the oldest.

The fundamental algorithm for the prediction model is LSTM. LSTM is an improved algorithm compared to Recurrent neural networks (RNNs). Moreover, LSTM is better at memorizing input data, which is emerged past than RNNs [24], [25].

A. THE CONCEPT OF LSTM

LSTM is an improved version of conventional RNNs that deals with the problem of long-term dependencies [26], [27]. So far, the simplest LSTM model is based on forward pass LSTM networks. In the forward pass LSTM networks, the h_t takes the input sequences forward (from the less recent timestamp to the most recent ones). Thus, the backward pass LSTM networks consider input sequences with the opposite direction, from the most recent to the less recent ones (backward direction) [24], [28]. The LSTM is capable of removing or adding information to the cell state. A module of LSTM networks can be categorized in three multiplicative gates: *forget gate*, *input gate*, and *output gate*. The forget gate consists of two inputs, h_{t-1} and x_t that are fed into the cell state C_{t-1} through sigmoid f_t. This gate determines whether the information is to be remembered or not. The h_t keeps all the useful information from past to time t. Because it goes through sigmoid, the output is placed between 0 and 1. If the output value is closer to 1, it is more likely to keep the inputs. The second, the input gate, decides to update cell state C_{t-1} with input x_t. With new candidate values $\widetilde{C}_t$ from the tanh layer, the output from the input gate updates the cell state. The last, the output gate, calculates what will be the output. Based on the cell state, the gate filters via sigmoid o_t. Formally, the formulas in a module of LSTM networks at time t are stated below [24], [28], [29]:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{2}$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{3}$$

$$h_t = o_t \odot tanh(c_t) \tag{4}$$

$$\widetilde{C}_t = tanh(W_c h_{t-1} + U_c x_t + b_c) \tag{5}$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{6}$$

Since all the formulas are at the time t, t-1 indicates that prior time sequence. Also, x_t is the input vector. As it is stated above, f_t, i_t, and o_t are for the forget, input, and output gates respectively. All U and W are the weight matrices, and all b denotes the bias vectors for each gate. The σ is the logistic sigmoid function that is applied component-wise [28].

The fact that LSTM networks can remember the information from a long time ago makes them *memory cells*. In the view of a module in LSTM networks, the new output cell state C_t is generated through multiplied previous cell state C_{t-1} with forget gates to decide the old states and input gates to determine how much each state is updated [28], [29].

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{7}$$

The bi-directional LSTM networks introduced by Schuster *et al.* take both past and future information [30]. The bi-directional LSTM is relatively advantageous regardless of tasks and contexts because it exploits the sequential information in both directions (from past to future and vice versa) [28], [31]. Besides, this paper utilizes an additional model of LSTM architecture to deal with general sequence to sequence problems. In particular, the encoder layer (the first LSTM layer) maps a variable-length input sequence to a representation that is produced in a fixed dimension vector with the learned information. Then, the decoder layer (the second LSTM layer) generates a variable-length target sequence through the vector representations. In this procedure, the decoder reconstructs the target time-series through the current hidden state and the prediction results based on previous time-step values. The LSTM encoder-decoder model has been applied successfully in various fields such as language translation and anomaly detection [32], [33].

B. WORD EMBEDDINGS

In machine learning, there are many ways to help treat text during modeling for natural language processing, such as document classification, sentiment analysis, and question-answering. Treating single words as unique symbols considers similarities and dissimilarities between words [34]. The easiest way to associate a representation with a text or a bundle of text is to choose a representation randomly. However, we cannot capture the semantic relationship between representations in this way. For this reason, many works propose to represent words as dense vectors with neural-network language modeling as training methods. This methodology is named 'Neural Embeddings' or 'Word Embeddings'. Word embedding is an algorithm to convert texts or words to vectors as representations with relations among them, such as capturing context, similarity, and semantic. When people train a machine learning model with the text or character types of data, it is necessary to convert it to a form that computers can understand.

Hence, people try to transform the data into numbers or vectors. For instance, 'King' turns to 1, 'Queen' maps to 2, 'Man' connects to 3, and 'Woman' converts to 4. One other way, one-hot encoding, is associating every integer number of the words with a binary vector that just one element is 1. The problem is that the represented numbers or vectors cannot contain a specific semantic or relationship, such as a relationship between 'King' - 'Queen' and 'Man' - 'Woman'. Even though the representations consist of vectors by one-hot encoding, the vectors cannot express connection and correlation among the data. Moreover, the more words in the dataset, the sparser and higher dimensions of vectors exist. Word embedding can solve those weak points.

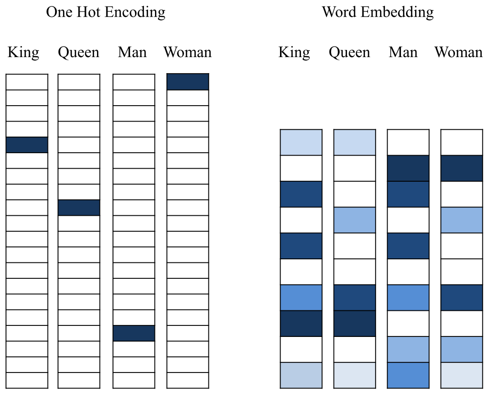


FIGURE 3. **Comparison of representations between one hot encoding and word embeddings.**

Word embedding has an abstract space of N dimensions to represent words to vectors. If there are words that have some correlations or e similar contexts between the terms, the representations (vectors) are close to each other. In Figure 3, the example compares two methods. A list of the rectangle is a symbolized vector of the label. If the color in a rectangle is getting darker, the real value in the vector is getting close to 1. Therefore, we can get words that appear in similar contexts through a cosine similarity measure in the vector space with word embeddings. Moreover, the word embedding method can adjust the number of dimensions that helps to have dense representations, and the representations are reliable and flexible because they learn from actual data. Because of the benefits, people exploit this in various natural language processing (NLP) tasks [35]. Besides, word embeddings are for general natural language processing and heavily customized or specific tasks. It means the spaces of the word embeddings can vary depending on the datasets.

In our work, we apply word embedding on *Application Name* in order to represent the *Application Name* data in latent vectors. The primary input data for the models is logs of app usage data. The input data of the LSTM algorithm is the *Application Name* attribute, which is a string. However, there is a common problem in the form of input for LSTM. The LSTM model, like all other neural networks, does not take raw texts as inputs. LSTMs are compatible only with numerical data as inputs. Therefore, the fundamental step is converting text values of *Application Name* to numerical values.

In this paper, the first layer of the LSTM model uses the Word Embeddings method for the inputs of *Application Name* of the models. The Word Embeddings help to represent the data in vectors with latent low dimensionalities. All application names are converted into dense vectors. Each application name corresponds to a word, and all application log data are a data set of the words such as documents. The apps that have similar contexts and there are relationships between them are represented with closer word embeddings.

In summary, this work tries to understand our data better and get deeper insights through data analysis. Then, this work tries to implement a reliable prediction model using LSTM-based architectures and Word Embeddings to predict next app by exploiting the app usage sequences.

C. PREDICTION MODEL

With the strong points of LSTM, the input of the LSTM algorithm is app usage sequences. As we presented before, there are three types of LSTM networks (forward, backward, and bi-directional) based on the direction that LSTM takes the inputs. Hence, we will implement those three LSTM models and compare them. Also, we define a sequence within different time windows: 5 minutes, 1 hour, 12 hours, one day, and one week. The reasons for deciding to use those time windows will be explained in Section IV and V. Then, we will train models with sequences within the time windows and try to evaluate the results. Our approach uses the embedding layer of the LSTM to find the word embeddings of the input data. After that, we train the LSTM, and we get the output through a softmax function. The output will be an array that contains probabilities of each app, meaning that the app with a higher likelihood is more likely to be the next app. In particular, given a sequence of n ordered apps $(a_{r-n}, a_{r-n+1}, \ldots, a_{r-1})$, the objective of the WhatsNextApp model in the training procedure is to minimize the sparse cross entropy:

$$-\frac{1}{D}\sum_{r=n}^{D} y_{temp_r} \log P_r(\hat{a}_r | a_{r-n}, a_{r-n+1}, \ldots, a_{r-1}) \quad (8)$$

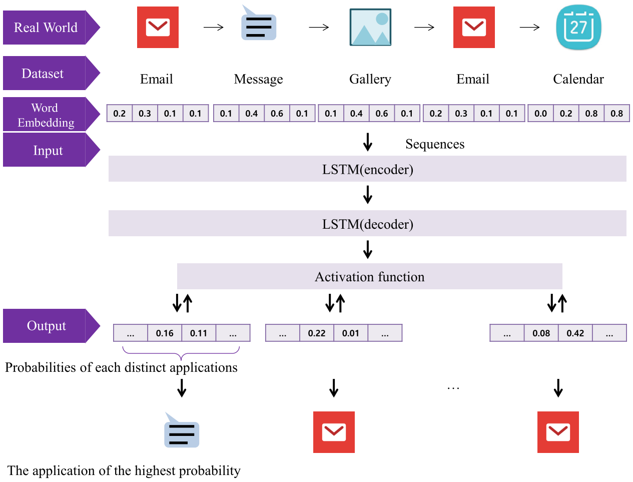


FIGURE 4. Overview of the prediction model (encoder-decoder).

where D is the set of all sequences, a multiclass classifier typically handles in the prediction task. At the prediction time, we employ softmax as activation, and we have:

$$P_r(\widehat{temp}_r | a_{r-n}, a_{r-n+1}, \ldots, a_{r-1}) = \frac{\exp(y_{\hat{a}_r})}{\sum_i \exp(y_{\hat{a}_i})} \quad (9)$$

This work also differentiates from the previous works as it addresses the problem of forecasting the next apps using time information of the app usage. As a result, the second part of our work is forecasting the next apps with an LSTM encoder-decoder. Figure 4 shows an overview of the model. The forecasting of the next apps with a specific time window includes the decoder that generating the next outputs repeatedly using their previous outputs. As a result, we can get several predicted apps if we implement the decoder that generates outputs sequentially. Each element of sequences includes the probabilities of each app to be the next app.

The main aim is to get a model that predicts the next app usage correctly. Therefore, the evaluation step focuses on estimating the predictive model performance. Before the evaluation, this research plans to obtain the top k apps by selecting the apps with the k highest probabilities. Then, the performance quality is assessed with the recall metric. The recall is a reliable metric, as it is used widely for similar purposes in state-of-the-art works [3], [11], [36], [37]. In the related work, k has values between 5 and 20. Therefore, we take the median value eight as k. In summary, we measure the recall at top-k (Recall@k, $k = 1, 2, 3, 4, 5, 6, 7$, and 8). When the predicted app is in the list of top-k apps, the recall is higher. When a prediction of app corresponds to smaller k, it can be interpreted as a better prediction.

Besides, the LSTM algorithm has several types. Therefore, we will compare the three LSTM networks models (forward pass LSTM, backward pass LSTM, and bidirectional LSTM) to identify a better quality of the prediction model within different time window sizes. Lastly, we will set the conditions of testing the models which are similar to other studies and assess the capability of our models. This evaluation helps to understand that our models can predict better than other studies. The time windows we created of 5 minutes, an hour, one day, and one week contain specific apps extended in chronological order. With these sequences of apps from different users, we train our network. New users have very little data about their usage app history, creating the user cold-start problem. Therefore, when new users appear, the sequences of apps within the specific windows are matched with the sequences of existing users that show similar patterns. Thus, based on the common patterns with existing users, the LSTM network can predict the next apps of new users with high accuracy. The LSTM network, including the word embedding layer, contributes to the high prediction accuracy as it exploits the sequential nature of the data. As explained in a previous section, the word embedding layer creates word embeddings with a shorter distance between them for app names that have a higher probability of appearing in adjacent positions in the sequence of apps.

We have a high-quality dataset with data from the TYDR app over nine months from 975 users and 19485 different apps. This guarantees a diverse dataset that can capture many different user behaviors in app usage and contribute to a good training of the LSTM network. In particular, we have created sequences of apps with specific time windows chronologically ordered that show the time dependencies between app usages. So, they are treated similarly to time series. After this feature extraction step, the first layer of the LSTM network is the embedding layer that, after the training, creates word embeddings where geometrical relationships between the app names represent semantic relationships between them. Apps with a smaller distance between them show a stronger relationship and a higher probability of appearing in neighbor positions in the sequences. Then, we have the LSTM layer that exploits the time dependencies of the app in the sequences to predict the next app accordingly. In LSTMs, the information flows through cell states that decide what apps in the sequences are essential for predicting the next app by giving them the proper weights. Also, as the LSTM network is trained, latent relationships between the different app sequences are captured, enhancing the higher prediction accuracy. So, when we have a sequence of apps from a new user, the LSTM network can use patterns from similar app sequences from other users and the latent representations between app sequences to predict the next app for the new user.

IV. DATA SET AND DATA ANALYSIS

A. DATA COLLECTION

We collected app event logs from Android smartphones with the app TYDR (Track Your Daily Routine) [38]. TYDR was designed for the collection of context data in combination with posing psychometric personality questionnaires [39], [40]. Collecting data with TYDR was approved by the ethics committee of the Technical University Berlin [41]. While a multitude of different context data was collected within the TYDR project, only the app event logs directly recorded by the Android operating system are used for this paper.

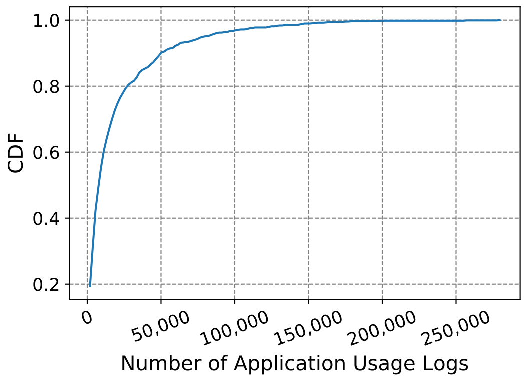


FIGURE 5. **CDF of app usage logs.**

We collected data in the nine months from October 2018 to June 2019. Overall, our data set has 1,166 users with 58,223,360 app event log entries. An anonymized user ID is stored for each entry, along with the application name, timestamp, and event type.

B. DATA CLEANING

In WhatsNextApp, we only used foreground events, i.e., apps that were actively used. Users could freely choose for how long they used TYDR. In the process of data cleaning, we removed outliers meaning users with very little data. More specifically, we checked the Cumulative Distributed Functions (CDFs) of the app usage logs in Figures 5, 6, 7.

We observed that they are shaped logarithmic, and the graphs sharply increase around the starting points. There is an extreme increment at 0.2 on the CDF axis. Therefore, we decided to use this 20% point as the reference for removing outliers. The other 80% have 1,963 app usage log entries, seven days of recording, and 40 different applications. This action removes 190 users. We removed one more user with timestamps from the future, probably from manually changing the time on his/her phone.

The final data set contains 22,037,158 million app usage log entries from 975 users. Overall, this data set contains 19,485 different apps. Note that we consider the package name string provided by the Android system[1] as the relevant source of the recorded event and define it as the app for the predictions.

C. ANALYSIS OF APPLICATION USAGE

This section gives insights about our data set for the apps' diversity and which ones are used most. Figure 8 shows the normalized density of the number of distinct applications used. The density is highest for the range of 50 to 75 different apps used. The range from 50 to 125 contains half of the users. With about 19,485 different apps in our data set, we assume

---

[1] https://developer.android.com/reference/android/app/usage/UsageEvents.Event#getPackageName(); accessed 2021-09-29

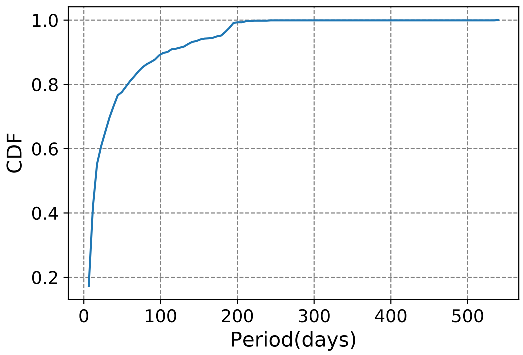


FIGURE 6. CDF of usage period.

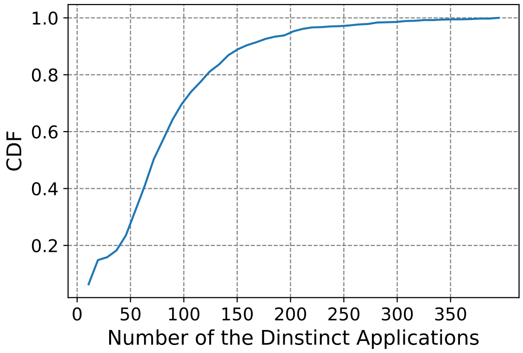


FIGURE 7. CDF of distinct used apps.

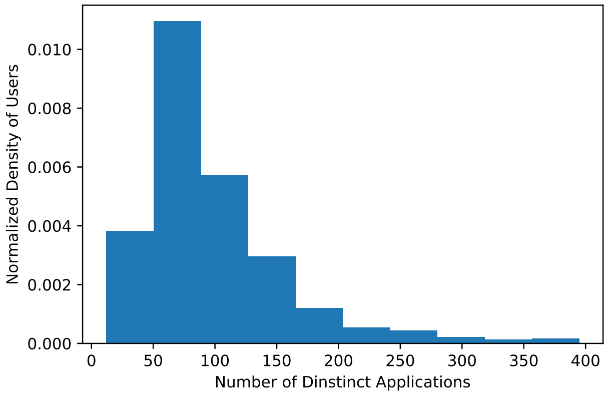


FIGURE 8. Normalized density with the number of the distinct apps.

that the users' overlap is not very large. WhatsApp is by far the most used app in our dataset, cf. Figure 9. Its foreground events account for nearly 20% of all application event logs of all 975 users. In the figure, we show the top 30 apps and their percentage of all app usage logs. Rank 13 represents less than 1% of all app use events, and rank 26 represents less than 0.5%. Overall, we observe a clear dominance of Social Networking Services (SNS) and messaging apps. The phone call and contact apps are also frequently used by users. Youtube and Spotify are the most frequently used entertainment-related apps in our data set.

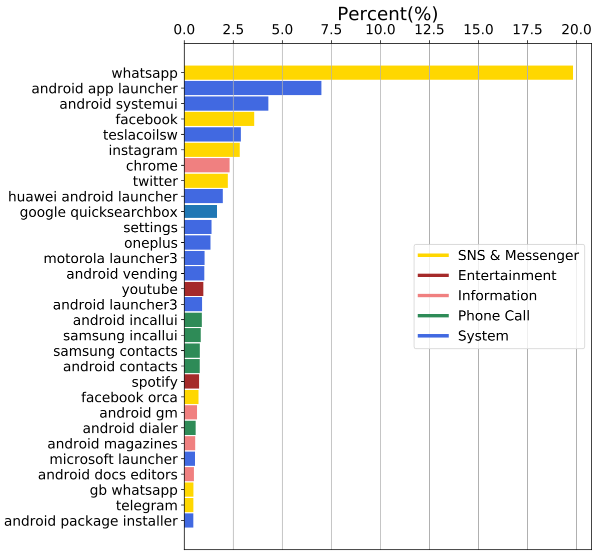


FIGURE 9. Top 30 apps and their percentage of the whole app usage over our dataset.

V. EXPERIMENTAL RESULTS

A. EXPERIMENT SETUP

Figure 10 shows an overview of the implementation. The implementation consists of three parts: pre-processing, model implementation, and evaluation.

1) PRE-PROCESSING

The pre-processing step is about preparing the variables that can affect the model implementation. According to the design, we take the app usage logs as inputs. Afterward, we split the dataset into training and testing sets. As WhatsNextApp is a general model, having separate users in training and testing sets is meaningful. Subsequently: 90% of the users for the training and 10% of the users for testing without intersections.

There are numerous choices of how much app usage log data we should include in a sequence. Several studies connect app usage with temporal features [5], [7], [9], [12], [22]. Temporal features are features that change over time. Therefore, the temporal characteristics are decisive for the optimal sequence length. This paper proposes that one sequence should contain the median number of the users' app usage logs per week. After that, it will be implemented with the median numbers of the app usage logs per day, 12 hours, 1 hour, and 5 minutes to compare the model's qualities depending on the life cycle periods and find the best time window. Previously, we fixed the number of app usage logs according to the periods. However, just binding the number of logs in one sequence is not accurate. The app usage logs have to be extracted relied on the periods through disposal and padding. Some users use smartphones more actively than the average user, and others have different app usage logs during the same period. Table 2 shows the different sequences of apps in fixed periods and the corresponding number of samples. In order to change the imbalanced situation, the logs

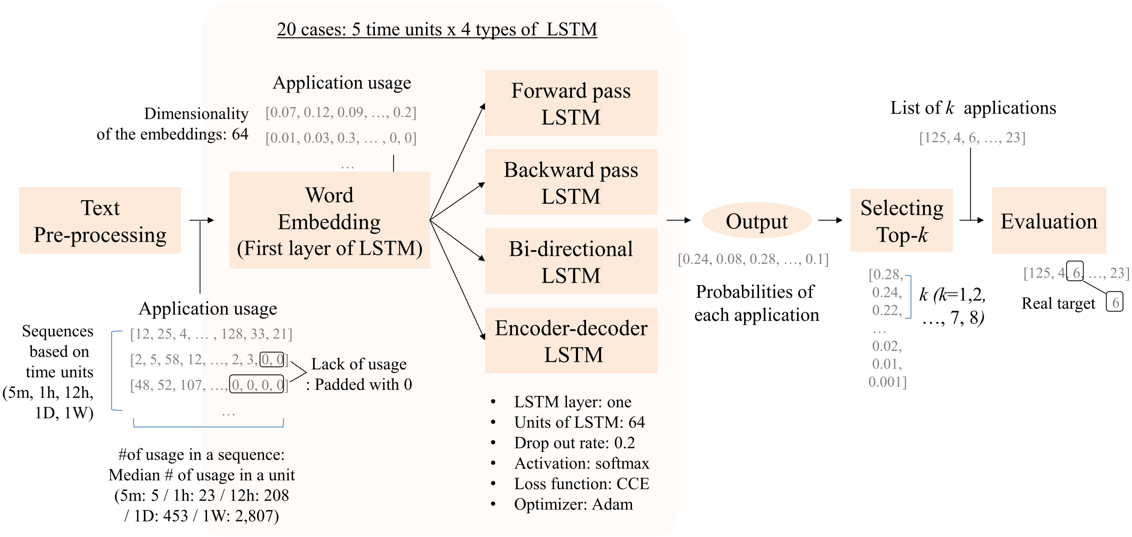


FIGURE 10. Overview of the WhatsNextApp implementation.

TABLE 2. Details of the sequences.

Unit	Apps per Sequence	Number of Samples	
		training	testing
5 minutes	5	22,18,741	250,125
1 hour	23	529,803	56,865
12 hours	208	71,668	7,488
1 Day	453	36,553	3,832
1 Week	2,807	6,114	642

in a sequence are truncated or padded with 0. Subsequently, we notice that the app usage logs are imbalanced regarding the number of times each app appears. Therefore, we try to address the next app prediction as a multiclass classification problem. Under these circumstances, a study by Han *et al.* suggests additional methods like under-sampling or oversampling to deal with the problem of imbalanced classes [42]. We planned to solve the imbalanced app usage data by using either the method of oversampling or under-sampling because a balanced data set improves the detection rate of minority classes in our models [42], [43]. We do random under-sampling as it consumes less computational resources than oversampling.

2) SETUP OF THE ARCHITECTURE

The first model is implemented for the prediction of the next app. When we feed the inputs of the LSTM model, the first layer generates the word embeddings. The text format of the *app name* is already tokenized in the pre-processing step. The embedding dimensionality is set to 64, the number of neurons is 64, and the dropout rate is 0.2. We use the softmax activation function, and finally, as loss function, we use the sparse cross-entropy. As an optimizer, we use the Adam optimizer, and we set the number of epochs to 20 [44] and the batch size to 8. To explain concretely about the model, it is crucial to figure out how the architecture of LSTM model is formed. The visualized architecture is depicted in Figure 11. In principle, the model takes the inputs, which are the productions of the previous phase. Also, the model has two layers: embedding and LSTM layers. The layers save the weights for discovering a collection of values to make inputs match the real target. Then, if the model completes the learning, it can get the ideal weights to make the predictions the most closely or correct with the real targets. In the middle of the structure, there are other components such as *Dropout, Dense, and Optimizer*. When the inputs are inserted, they encounter the first layer, which conducts the word embeddings. The text format of the *Application name* is already tokenized in the previous pre-processing. Then, the vectors of every application are extracted that contain semantic distance among associated apps. This layer is set as the first layer of the LSTM network. Through the embedding layer, every

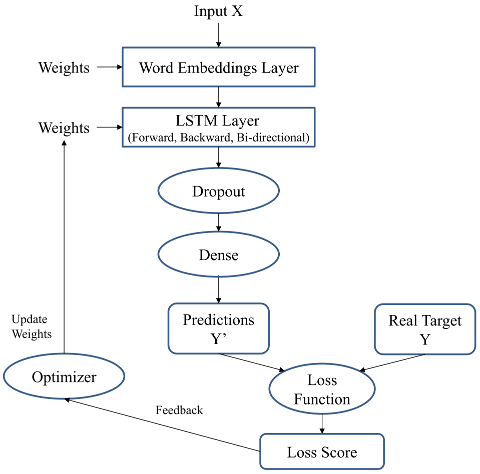


FIGURE 11. Architecture of the LSTM model.

token which stands for the distinct application names are transformed into dense vectors included correlations between them. In this layer, random vectors are assigned at the start; embedding vectors are generated by learning the weights. This layer accepts two-dimensional inputs (samples, length of a sequence). The weights are assigned initially and continuously adjusted through the training. The outputs formulate a three-dimensional shape (samples, length of a sequence, the dimensionality of the embeddings) [45].

In this work, the embedding dimension is 64. The way of tuning the number of dimensionalities is described in similar research. According to [3], the result of prediction is improved when the embedding size is getting bigger. However, when the embedding size exceeds 50, the growth is slowed down, and there is only 5% less deviation than when the embedding size is 600. Moreover, the execution time is exponentially higher when the embedding size is over 200. Hence, considering two factors, this research decides to try a similar number of 50.

After the first layer, the transformed inputs as vectors meet the second layer, the LSTM layer. We implemented three types of LSTM networks: forward (default), backward, and bi-directional in terms of the direction of passing the inputs. There will be 15 different models because we already have five types of time windows for the input sequences with three categories of LSTM models. We define the number of units, 64, which indicates the number of neurons or nodes of the LSTM layer. If the number increases, it will contribute to higher model quality. However, the execution performance is getting worse [45]. We tried to tune the number for the units with the users' inputs for the training. However, the execution environment, precisely the size of memory, cannot afford more than 64. Then, the procedure of *Dropout* is implemented for the LSTM network in order to avoid the overfitting [46]. One of the cases is when a model is working with a training dataset almost correctly, like memorizing the entire dataset. However, it has a deficiency in dealing with new data. The main idea is dropping out (zeroed out) units randomly in a network that disconnects incoming and outgoing connections [46]. We can customize the dropout rate, which means the fraction. We fix the rate as 0.2 because the referred study [45] shows that the prevailing rate is set between 0.2 and 0.5. The *Dense*, working with softmax activation, means that it will generate an array of a designated number of probability scores. For our case, the number of the units for the *Dense* step is identical with the amount of the different apps, 19,485, because we want to get the probabilities of every app.

These LSTM networks also have tens of millions of weights that impact the predictions. The *loss function* calculates scores (distances) between the predictions (outputs) and the real targets, and it assesses how the networks are doing well for the inputs. In this implementation, categorical cross-entropy (CCE) is applied. According to the [47], CCE, which is eligible for cases of classifications of classes, shows more improved performances than the function of mean absolute error (MAE). The networks regard the loss scores as feedback indicators to modify the weights. The *optimizer* has the role of adjusting the scores. The job of the *optimizer* is to make the loss score lower through the backpropagation algorithm. Though the execution sets the weights of the LSTM networks with random values at first, the weights go in the direction of making the decline of the loss score during the training with the data [45]. We select an adaptive moment estimation (Adam) optimizer for this development. The Adam optimization estimates adaptive learning rates from the first and second moments of the gradients. We select Adam optimization because it is beneficial for the updates of massive parameters and memory consumption [48]. Besides, we have to use the technique of epoch and batches because we want to avoid model overfitting through iterative gradient descent works, and the app usage data is too big to process them at once. In the following epochs, the weights are adjusted in the networks. We set 20 epochs because in [44] Greff *et al.* stated that there were no changes after fifteen epochs without improvement of validation data. Also, we set the batch size to 8 to prevent the entire dataset from passing into the networks at once.

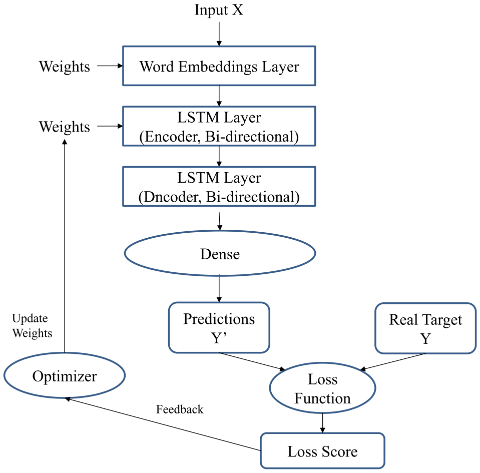


FIGURE 12. **Architecture of the LSTM model (encoder-decoder).**

3) MODEL FOR SEVERAL STEPS OF NEXT APPS

The second model can predict several apps, and it is presented in Figure 12. The first step is identical to the model for one next app. The tokenized inputs are formed into sequences, and the embedding layer receives the inputs. The weights start with a random value during the transformation and are repeatedly adjusted. The defined number of latent dimensionality is 64, indicating dimensions of compressed representations. The encoder returns its internal state that serves the next layer, the decoder. The implementation keeps information of states in variables. The decoder layer is trained to generate target sequences by adding the previous output to the subsequent input data and repeating the process many times. The important thing is that the initial state for the decoder is the encoder's state vectors. It is the way the decoder gains information for the networks. Therefore, the stored states variables of the encoder are used to set up the decoder's initial state. The size target sequence in the decoder is 1. State vectors and previously generated targets are inserted into the decoder for several prediction steps. The next prediction is based on the former prediction. We have to define the rule to stop the decoder loop here. The encoder-decoder model for other cases such as translation utilizes end-of-sequence (sentence) token to make the model stop predictions [49]. However, the app usage does not have some sort of tokens. Therefore, the model only predicts ten app usage as a sequence. The ten app usage is decided based on multiple reasons. Firstly, the app usage, which happens many steps later, is not helpful because the app usage bursts unexpectedly happen even in 10 seconds. The second reason is that the model could predict different results when the new app usage appears. The other reason is that the model can fully cover the 5 minutes of future usage because the median number of app usage in 5 minutes is 5. The rest of the parts, such as dense, loss function, optimization, epochs, and batches, correspond to the first model.

4) RECALL@8

In the tests, we use the recall metric to measure the prediction performance. First, we calculate the score of recall for the top k apps with the highest probabilities to be the next app, and we name the score recall@k. We give the definition of recall@k in Equation 10, as shown at the bottom of the page.

We denote D^{test} set of test data, a_i is the real target app of ith test data, $Apps_i$ is predicted k apps, and I is an indicator function [3], [36], [50].

The recall indicates the proportions of accurately predicted apps out of the number of apps in the foreground in the testing set. The recall metric is the most frequently used assessment in recent studies [2], [3], [8], [9], [11], [12], [22]. Unfortunately, even though they use the identical method to calculate their result, they name the same metric differently: hit rate, recall, and accuracy. However, we verified that all the formulas are the same, and the basis of them is [50], which evaluates the performance of top-k recommendations.

When we generate a sequence in the implementation section, within a time window, we obtain the median number of app usage for each time window, and the median number of logs formulates a sequence. In the experiments, we try five different time windows: 5 minutes with five usage events per sequence, 1 hour with 23, 12 hours with 208, 1 day with 453, and 1 week with 2,807, and we use three different LSTM architectures. Also, this work uses a standard method, MFU (Most Frequently Used), as a baseline, also presented in similar works [2], [3], [37].

B. NEXT APP PREDICTION

1) COLD-START PROBLEM – MODEL FOR NEW USERS

The first prediction model is for predicting the next app. The performance of this model is tested on users unseen in the training set, evaluating the performance for new users (cold-start problem). The model's objective is to find which LSTM model has a better prediction performance in terms of recall@k and within which time window. The LSTM models are forward, backward, and bi-directional. Firstly, we want to check recall@k for the 500 most frequent apps. As we mentioned in the Data Analysis section, the number of different apps is 19,485. It is the largest number of distinct apps compared to other studies in Table 1. Our amount of specific apps is about two times more than the biggest number [11]. Therefore, we select 500 different apps, then calculate the recall@8 for the 500 different apps. In the testing set, only apps that are relevant to the 500 different apps comprise D^{test} and a_i to calculate the recall. Table 3 includes the recall@k of the bi-directional LSTM and MFU model. When the value of the k parameter is 8, and the time window is 1 hour, we obtain the highest value for recall@k: 82.74%. A limitation of LSTM is that it reduces the prediction accuracy for very long app sequences since they face the problem of vanishing gradients and exploding gradients [26]. For example, we notice that our accuracy in the cold-start problem for a 1 hour time window when using the 500 most frequent apps is 82.74%. The prediction accuracy decreases as the app sequences become more prolonged, with 70.14% accuracy for a one-day time window and 67.50% accuracy for a one-week window. For a time window of 12 hours, the accuracy decreases slightly compared to the one-hour window, and when we have very short sequences (for a time window of 5 minutes), the accuracy is again worse, 77.37% because we fail to represent the sequential relationship of the apps adequately. One solution to the problem of vanishing gradients and exploding gradients is to use an attention mechanism like temporal attention that gives different weights to the apps within the sequence.

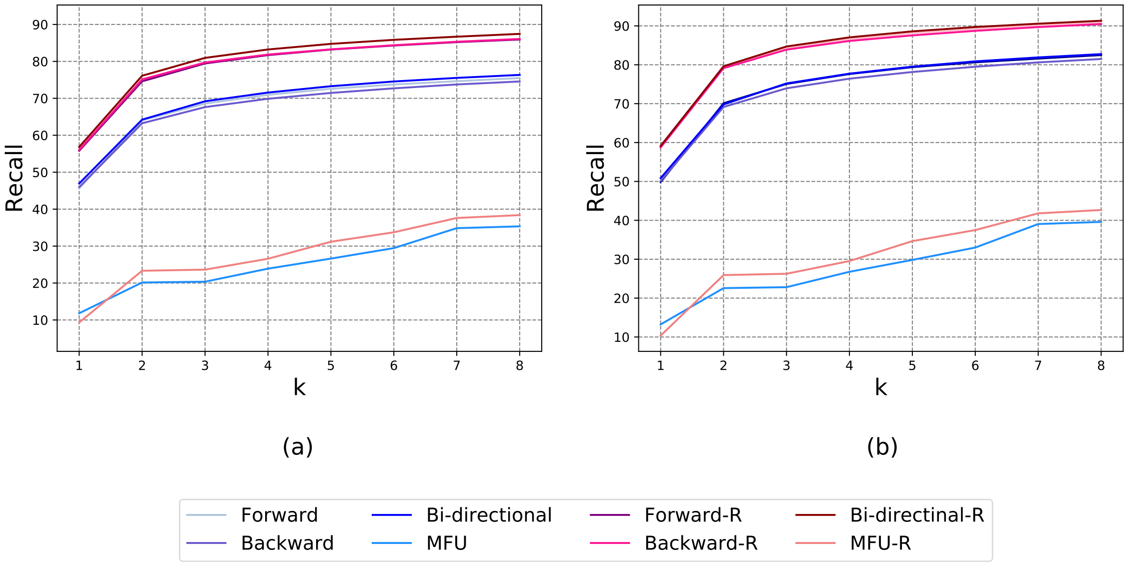
2) COMPARISON WITH RELATED WORK – MODEL FOR EXISTING USERS

All other works in Table 1 utilize sequences of the app by using data from the same users both in training and testing sets. Therefore, we decide to adopt the same procedure and compare the results. We use 90% of app usage data from all users for the training set and 10% of app usage data from all users for the testing set. Then, we evaluate the results of our models within the best time window, 1 hour. Figure 13 depicts recall@k with the original testing set divided by app usage. We notice that the model with the randomly selected testing

$$recall@k = \frac{\sum_{i=1}^{|D^{test}|} I(a_i, Apps_i)}{|D^{test}|}, \quad I = \begin{cases} 1 & a_i \in Apps_i \\ 0 & otherwise \end{cases} \quad k = 1, \ldots, 8 \tag{10}$$

TABLE 3. **Recall@*k* of the LSTM (bi-directional) and MFU models about the 500 most frequent apps for the cold-start problem.**

	LSTM (bi-directional)						MFU				
k	5m	1h	12h	1D	1W	*k*	5m	1h	12h	1D	1W
1	47.37	50.86	45.11	38.13	36.94	1	16.53	13.23	12.25	9.51	8.35
2	61.50	69.80	65.11	56.80	49.91	2	22.96	22.57	22.11	19.46	17.41
3	66.71	75.21	70.62	61.71	54.53	3	23.45	22.81	22.38	19.81	17.58
4	69.79	77.72	73.75	64.57	58.44	4	28.39	26.77	25.93	23.49	20.25
5	72.24	79.52	75.41	66.59	61.28	5	30.75	29.82	29.17	26.91	24.51
6	74.20	80.88	76.77	67.88	63.41	6	35.18	33.00	31.87	29.70	27.18
7	76.14	81.90	78.11	69.10	65.19	7	42.93	39.05	37.26	36.09	33.39
8	77.37	**82.74**	79.29	70.14	67.50	8	**43.87**	39.59	37.71	36.68	33.75



(a)

(b)

FIGURE 13. **Recall@*k* of all the models with testing data set (model for existing users; users can be in training and testing set; time unit: 1h), (a) about all app and (b) about 500 apps, "R" stands for with randomly selected testing data set of all users.**

data of all users can predict the next app more precisely in terms of recall@k both for the 500 most frequent apps (Figure 13 (b)) but also all apps (Figure 13 (a)). When using the 500 most frequent apps, we achieve 92% of recall@*8*.

3) COMPARISON WITH RELATED WORK – A MODEL FOR ALL APPS FOR NEW USERS

As a next step, we again run the experiments for all the different apps (19,485) for new users (cold-start problem) to further validate our model's efficiency. First of all, the best model with a recall value of 76.34% is the bi-directional LSTM model, which predicts eight apps with 1 hour time window. When we see Figure 14, we can figure out the trends of the models with 1 hour time window. There is only a slight deviation in the three LSTM models. However, the recall@*k* of the MFU model is less than half of the three LSTM models in broad outlines. Although the recall scores of the three LSTM models are not entirely different, always the bi-directional LSTM model is better than the others.

Especially when the *k* is getting smaller, the forward and backward LSTM models with 12 hours, one-day, and one-week time windows predict poorly, as we can see in Tables 4 and 5. Some predictions of the forward and backward LSTM models (when *k* is 1, and the time window is 12 hours) are worse than the MFU model. The bi-directional LSTM model achieves over 32% in any condition. Thus, the model which can predict the next app with overall *k* is the bi-directional LSTM model. In all three models, the 1 hour time window generates the most outstanding recall scores among all-time windows. The LSTM models generally accomplish over 45% of recall score in all conditions even though *k* is 1. However, 12 hours, one-day, and one-week time windows achieve relatively low recall scores when *k* is small. Besides, in the situation that *k* increases, Recall@*k* of those three windows is consistently lower than the 1 hour time window. In Figure 15, there is a trend curve to compare the time windows of the bi-directional model. The time window of 12 hours also seems to perform well; however, it does not predict well enough when k is 1. Besides, the 5-minute model

TABLE 4. Recall@*k* of the LSTM (forward) and LSTM (backward) models for all the apps (19,485 apps) and for the cold-start problem.

	LSTM(forward)						LSTM(backward)				
k	5m	1h	12h	1D	1W	*k*	5m	1h	12h	1D	1W
1	43.39	45.63	8.83	8.82	7.94	1	42.84	45.98	8.83	8.82	7.94
2	55.38	64.04	19.79	11.85	15.26	2	54.77	63.25	19.79	17.25	11.68
3	60.24	68.58	22.69	20.28	21.50	3	59.55	67.63	22.69	24.11	19.00
4	63.02	70.95	29.01	27.14	25.23	4	62.86	69.88	29.01	27.14	26.64
5	65.40	72.59	30.53	32.80	32.87	5	65.05	71.46	33.83	33.04	32.87
6	67.47	73.75	34.79	38.70	38.32	6	66.87	72.71	38.09	38.70	35.20
7	68.91	74.65	39.61	39.22	40.65	7	68.41	73.75	41.27	40.55	40.65
8	70.20	**75.49**	42.03	41.08	42.99	8	69.51	**74.56**	41.80	43.82	42.99

TABLE 5. Recall@*k* of the LSTM (bi-directional) and MFU models for all the apps (19,485 apps) and for the cold-start problem.

	LSTM(bi-directional)						MFU				
k	5m	1h	12h	1D	1W	*k*	5m	1h	12h	1D	1W
1	43.60	46.94	40.63	36.46	32.40	1	14.62	11.81	10.96	8.43	7.32
2	56.30	64.23	58.79	54.31	43.77	2	20.31	20.14	19.79	17.25	15.26
3	61.06	69.20	63.88	59.03	47.82	3	20.74	20.36	20.03	17.56	15.42
4	63.85	71.56	66.77	61.77	51.25	4	25.11	23.89	23.21	20.82	17.76
5	66.05	73.28	68.34	63.70	53.89	5	27.20	26.62	26.11	23.85	21.50
6	67.83	74.56	69.63	64.93	55.76	6	31.11	29.45	28.53	26.33	23.83
7	69.59	75.54	70.83	66.10	57.32	7	37.97	34.86	33.35	31.99	29.28
8	70.70	**76.34**	71.96	67.09	59.35	8	**38.80**	35.34	33.75	32.52	29.60

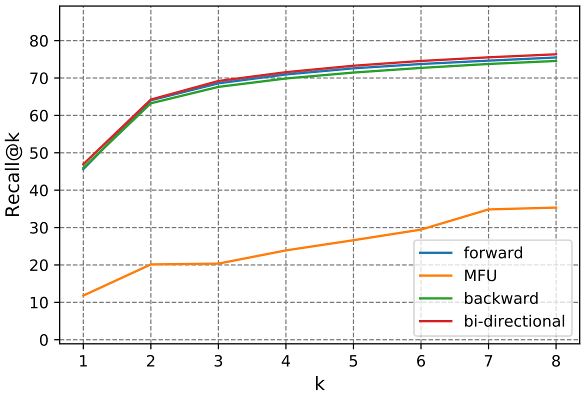


FIGURE 14. Recall@*k* of all models for all the apps (19,485 apps) for the cold-start problem and time window: 1h.

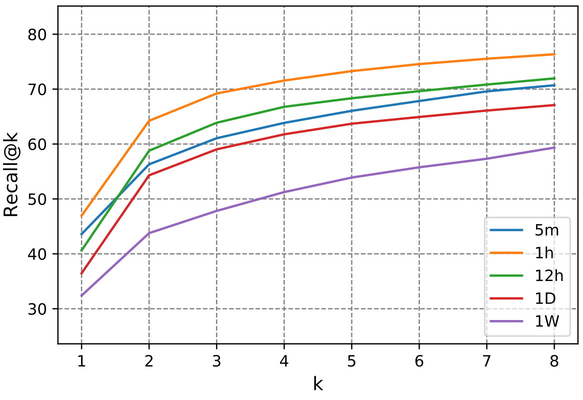


FIGURE 15. Recall@*k* of the bi-directional LSTM model for all the apps (19,485 apps) and for the cold-start problem.

is also more reliable than the one-day and one-week ones, and it has no poor quality compared to the 1 hour and 12 hours models. The 5-minute model is also suitable for apps such as pre-loading because the model can work with data for a short period with a small amount of computing power. Hence, considering which time window is most helpful in predicting performance, the 1-hour time window is the right one. In other words, the app usage during 1 hour is the most advantageous pattern to predict the next app.

C. PREDICTION OF THE NEXT TEN APPS

Next, we implement the model of encoder-decoder to predict app usage in the short term. The prediction result is a sequence of apps that a user will likely open (in the foreground) consecutively. We define the steps of predicted apps as 10. From a practical point of view, excessive numbers of predicted apps are disadvantageous for pre-loading. Moreover, a user's frequent new logs can change the results. We notice that the 1-hour time window gives better results for one and several next time steps of app prediction. In particular, the larger the *k*, the better the recall value.

Figure 16 shows the recall@*8* of the encoder-decoder model with 500 apps for the next ten apps. Within all time windows, we accomplish a recall of over 40% at all steps. We get the best results with the 1-hour time window. So we

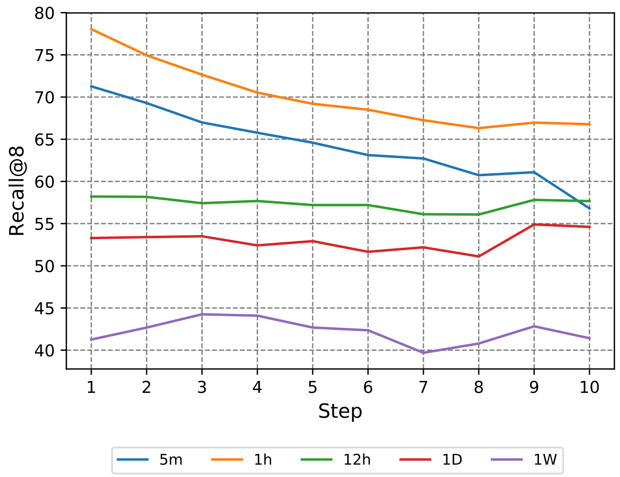


FIGURE 16. The trend of the encoder-decoder model depended on different time units (recall@8, 500 apps).

can see that for predicting the next app, we get 70.37% of recall@8, and for predicting the next two apps, 67.61%.

VI. CONCLUSION AND FUTURE WORK

This paper presented WhatsNextApp, a deep learning approach for next app prediction that utilizes LSTM-based models. The expected advantages of accurate next app prediction are (a) UX (user experience) optimizations, (b) loading apps that are about to be used into memory (or keeping them in memory), (c) optimize network infrastructure (as different apps produce different amounts of traffic with different endpoints). WhatsNextApp takes the temporal feature as an input, app usage sequences, and predicts the next app. The model is generic and uses supervised multiclass classification. We have a relatively large data set consisting of 975 users and 19,485 different apps. We take the 500 most frequently used apps, and for the cold-start problem, we reach for the 1-hour time window a recall@8 of 82.7%, whereas in the case of existing users, a recall@8 of 92%. Finally, we use an LSTM encoder-decoder to predict the next ten apps when dealing with new users. For the 1-hour time window, we achieve 70.37% of recall@8, whereas the recall@8 decreases for predicting the next two apps with a value of 67.61% and 65.32% for the three next apps.

Even though our results are promising, we use only the time sequence app usage as a feature, and we believe that we can further improve the results by extracting more temporal features like weekday and weekends patterns. Additionally, We could aim at predicting which app is going to be used next and when.

REFERENCES

[1] H. Cao and M. Lin, "Mining smartphone data for app usage prediction and recommendations: A survey," *Pervasive Mobile Comput.*, vol. 37, pp. 1–22, Jun. 2017.

[2] X. Chen, Y. Wang, J. He, S. Pan, Y. Li, and P. Zhang, "CAP: Context-aware app usage prediction with heterogeneous graph embedding," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 3, no. 1, pp. 1–25, Mar. 2019.

[3] S. Zhao, Z. Luo, Z. Jiang, H. Wang, F. Xu, S. Li, J. Yin, and G. Pan, "AppUsage2 Vec: Modeling smartphone app usage for prediction," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1322–1333.

[4] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, and T. Choudhury, "Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns," in *Proc. 17th Annu. Int. Symp. Int. Symp. Wearable Comput. (ISWC)*, 2013, pp. 69–76.

[5] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2012, pp. 113–126.

[6] C. V. N. Index, "Cisco visual networking index: Forecast and trends, 2017–2022," Cisco, San Jose, CA, USA, White Paper c11-74149, Feb. 2019.

[7] R. Baeza-Yates, D. Jiang, F. Silvestri, and B. Harrison, "Predicting the next app that you are going to use," in *Proc. 8th ACM Int. Conf. Web Search Data Mining*, Feb. 2015, pp. 285–294.

[8] X. Zou, W. Zhang, S. Li, and G. Pan, "Prophet: What app you wish to use next," in *Proc. ACM Conf. Pervasive Ubiquitous Comput. Adjunct Publication*, Sep. 2013, pp. 167–170.

[9] C. Sun, J. Zheng, H. Yao, Y. Wang, and D. F. Hsu, "AppRush: Using dynamic shortcuts to facilitate application launching on mobile devices," *Proc. Comput. Sci.*, vol. 19, pp. 445–452, Jan. 2013.

[10] M. Hashemi and J. Herbert, "A next application prediction service using the BaranC framework," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jun. 2016, pp. 519–523.

[11] N. Natarajan, D. Shin, and I. S. Dhillon, "Which app will you use next?: Collaborative filtering with interactional context," in *Proc. 7th ACM Conf. Recommender Syst.*, Oct. 2013, pp. 201–208.

[12] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, Sep. 2013, pp. 275–284.

[13] D. Han, J. Li, W. Li, R. Liu, and H. Chen, "An app usage recommender system: Improving prediction accuracy for both warm and cold start users," *Multimedia Syst.*, vol. 25, no. 6, pp. 603–616, Dec. 2019.

[14] D. Yu, Y. Li, F. Xu, P. Zhang, and V. Kostakos, "Smartphone app usage prediction using points of interest," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 1–21, Jan. 2018.

[15] C. Fang, Y. Wang, D. Mu, and Z. Wu, "Next-app prediction by fusing semantic information with sequential behavior," *IEEE Access*, vol. 6, pp. 73489–73498, 2018.

[16] C. Chen, T. Maekawa, A. Daichi, and T. Hara, "Preliminary investigation of predicting next-use mobile apps using app semantic representations," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 391–394.

[17] S. Lee, R. Ha, and H. Cha, "Click sequence prediction in Android mobile applications," *IEEE Trans. Human-Mach. Syst.*, vol. 49, no. 3, pp. 278–289, Jun. 2019.

[18] Y. Lee, S. Cho, and J. Choi, "App usage prediction for dual display device via two-phase sequence modeling," *Pervas. Mobile Comput.*, vol. 58, Aug. 2019, Art. no. 101025.

[19] Z. Shen, K. Yang, W. Du, X. Zhao, and J. Zou, "DeepAPP: A deep reinforcement learning framework for mobile application usage prediction," in *Proc. 17th Conf. Embedded Netw. Sensor Syst.*, Nov. 2019, pp. 153–165.

[20] Y. Jiang, X. Du, and T. Jin, "Using combined network information to predict mobile application usage," *Phys. A, Stat. Mech. Appl.*, vol. 515, pp. 430–439, Feb. 2019.

[21] T. M. T. Do and D. Gatica-Perez, "Where and what: Using smartphones to predict next locations and applications in daily life," *Pervasive Mobile Comput.*, vol. 12, pp. 79–91, Jun. 2014.

[22] K. Huang, C. Zhang, X. Ma, and G. Chen, "Predicting mobile application usage using contextual information," in *Proc. ACM Conf. Ubiquitous Comput. (UbiComp)*, 2012, pp. 1059–1065.

[23] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, Dec. 2000.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7, nos. 3–4, pp. 197–387, Jun. 2014.

[26] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[28] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, Aug. 2016, pp. 1064–1074.

[29] C. Olah, "Understanding LSTM networks," Colah's Blog, Tech. Rep., Aug. 2015. [Online]. Available: https://colah.github.io/

[30] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[31] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 4, Aug. 2005, pp. 2047–2052.

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 2. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112.

[33] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," in *Proc. ICML Anomaly Detection Workshop*, Jul. 2016.

[34] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2177–2185.

[35] D. Ganguly, D. Roy, M. Mitra, and G. J. F. Jones, "Word embedding based generalized language model for information retrieval," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 2015, pp. 795–798.

[36] Z.-X. Liao, S.-C. Li, W.-C. Peng, P. S. Yu, and T.-C. Liu, "On the feature discovery for app usage prediction in smartphones," in *Proc. IEEE 13th Int. Conf. Data Mining*, Dec. 2013, pp. 1127–1132.

[37] Z.-X. Liao, Y.-C. Pan, W.-C. Peng, and P.-R. Lei, "On mining mobile apps usage behavior for predicting apps usage in smartphones," in *Proc. 22nd ACM Int. Conf. Conf. Inf. Knowl. Manage. (CIKM)*, 2013, pp. 609–618.

[38] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, R. Pryss, and J. Zimmermann, "TYDR: Track your daily routine. Android app for tracking smartphone sensor and usage data," in *Proc. 5th Int. Conf. Mobile Softw. Eng. Syst.*, May 2018, pp. 72–75.

[39] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, R. Pryss, and J. Zimmermann, "Context data categories and privacy model for mobile data collection apps," *Proc. Comput. Sci.*, vol. 134, pp. 18–25, Jan. 2018.

[40] F. Beierle, T. Probst, M. Allemand, J. Zimmermann, R. Pryss, P. Neff, W. Schlee, S. Stieger, and S. Budimir, "Frequency and duration of daily smartphone usage in relation to personality traits," *Digit. Psychol.*, vol. 1, no. 1, pp. 20–28, Jun. 2020.

[41] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, J. Zimmermann, and R. Pryss, "What data are smartphone users willing to share with researchers?" *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 6, pp. 2277–2289, Jun. 2020.

[42] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new oversampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*, D.-S. Huang, X.-P. Zhang, and G.-B. Huang, Eds. Berlin, Germany: Springer, 2005, pp. 878–887.

[43] N. V. Chawla, N. Japkowicz, and A. Kolcz, "Editorial: Special issue on learning from imbalanced data sets," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 1–6, Jun. 2004.

[44] K. Greff, R. K. Srivastava, J. Koutnìk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[45] C. Francois, "Deep learning with PyThon," Manning Publications, Shelter Island, NY, USA, Tech. Rep., 2018.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2004.

[47] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8778–8788.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[49] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[50] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proc. 4th ACM Conf. Recommender Syst. (RecSys)*, 2010, pp. 39–46.

KATERINA KATSAROU received the Diploma degree from the Department of Electrical and Computer Engineering, Polytechnic School, University of Patras, Greece, and the master's degree in computer science from the University of Ioannina, Greece. In January 2018, she joined the Service-Centric Networking Group, Technische Universität Berlin, as a Research Scientist in data science and machine learning with a focus on natural language processing.

GEUNHYE YU received the B.A. degree in computer engineering from Hanbat National University in 2011, and the M.A. degree in computer science from Technische Universität Berlin in 2019. During her master's studies, she worked as a Research Assistant with the Fraunhofer Institute for Production Systems and Design Technology, Berlin, Germany. She is currently a Researcher in data science with the Korea Environment Institute.

FELIX BEIERLE (Member, IEEE) received the M.A. degree in media studies and American studies from the University of Marburg in 2009, the M.Sc. degree in computer science from the University of Hagen in 2014, and the Ph.D. degree in computer science from Technische Universität Berlin in 2020.

During his studies, he worked as a Software Engineer with Capgemini. He is currently a Postdoctoral Researcher in medical informatics with the Institute of Clinical Epidemiology and Biometry, University of Würzburg, Germany. His research interests include ubiquitous computing, social networking, recommender systems, and mHealth.

• • •