ATPP: A Mobile App Prediction System Based on Deep Marked Temporal Point Processes

KANG YANG, University of California, Merced, USA
XI ZHAO and JIANHUA ZOU, Xi'an Jiaotong University, China
WAN DU, University of California, Merced, USA

Predicting the next application (app) a user will open is essential for improving the user experience, e.g., app pre-loading and app recommendation. Unlike previous solutions that only predict which app the user will open, this article predicts both the next app and the time to open it. Time prediction is essential to avoid loading the next app too early and consuming unnecessary resources on smartphones. To predict the next app and open time jointly, we model the app usage sequence as a marked temporal point process (MTPP), whose conditional intensity function can capture the probability of a new app usage event. We develop a novel data-driven MTPP-based app prediction system, named ATPP (App Temporal Point Process), which adopts a recurrent neural network architecture to learn the MTPP conditional intensity function for app prediction. ATPP adopts a set of techniques to incorporate the unique features of app prediction in our RNN architecture, including learning the correlated usage behavior of different apps by representation learning, the temporal dependency of app usage by an attention mechanism, and the location-related app usage behavior by feature extraction and fusion layer. We conduct extensive experiments on a large-scale anonymized app usage dataset to verify ATPP's effectiveness.

CCS Concepts: • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools*; • **Computing methodologies** → *Neural networks;*

Additional Key Words and Phrases: Mobile devices, application usage prediction, marked temporal point process, recurrent neural networks

ACM Reference format:
Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. 2023. *ATPP*: A Mobile App Prediction System Based on Deep Marked Temporal Point Processes. *ACM Trans. Sensor Netw.* 19, 3, Article 71 (April 2023), 24 pages.
https://doi.org/10.1145/3582555

1 INTRODUCTION

In a 2015 survey [33], almost 36% of mobile users demanded that app loading time should be less than 2 seconds; 46% of iOS apps and 53% of Android apps take more than 2 seconds to load, resulting in poor user experience. An effective way to minimize loading time is to pre-load the next app into memory before a mobile user opens it [59]. Many app prediction works have been done;

Authors' addresses: K. Yang and W. Du (corresponding author), University of California, Merced, CA, 95340; emails: {kyang73, wdu3}@ucmerced.edu; X. Zhao and J. Zou, Xi'an Jiaotong University, Xi'an, China, 710049; emails: Zhaoxi1@mail.xjtu.edu.cn, jhzou@sei.xjtu.edu.cn.

1550-4859/2023/04-ART71 $15.00
https://doi.org/10.1145/3582555

however, they are mainly focused on the prediction of the next app but ignore the time that the user will open that app [6, 12, 26, 38, 43, 60]. Time prediction is important to efficiently pre-load the next predicted app into memory, because it may consume too much energy and memory on smartphones unnecessarily if the predicted app is loaded into memory too early [5].

This article proposes to jointly predict the next app and its open time by modeling the app usage behavior of a user as a **Marked Temporal Point Process (MTTP)** [1]. We learn an MTPP model with a conditional intensity function that quantifies the probability of a new app usage event conditioned by the previous app events. In this article, we implement two MTPP conditional intensity functions, i.e., the Hawkes process [23] and the Homogeneous Poisson process [1], to model app usage behavior. They have been widely used for time-series data prediction in many applications, such as human social activities [18] and online-user engagement [16]. However, based on our experiments on an anonymized app usage dataset of 443 users, such a simple MTPP-based method has poor performance. The major reason is that it is hard to find a proper MTPP intensity function to explicitly capture the influence of past app usage events and incorporate the unique features of app prediction, such as the correlated usage behavior of different apps, the temporal dependency of app usage, and the location-related app usage behavior.

To address the above limitations, we develop a novel data-driven MTPP-based app prediction system, named *ATPP* (App Temporal Point Process). It utilizes **Recurrent Neural Network (RNN)** [20] to learn the conditional intensity function of the MTPP model based on historical app usage data. We implement the RNN model by a set of **Gated Recurrent Units (GRUs)** [7], which transforms an app usage sequence into a sequence of hidden states. Each state is a feature vector learned via previous app usage events. Furthermore, *ATPP* proposes a novel RNN framework that incorporates the temporal dependency and spatial context of app usage events into app prediction. We design two feature extractors for temporal dependency and spatial context, respectively. The extracted temporal and spatial features are fused by the Hadamard product into the RNN framework for predicting the next app ID and its open time.

The temporal app usage pattern is that historical app usage events have different influences on the usage of the next app. Taking the usage sequence "Amazon, WhatsApp, ApplePay" as an example, a user receives a message from WhatsApp while she is using Amazon for online shopping. After she returns from WhatsApp, she pays the Amazon bill by ApplePay. During this process, the usage of ApplePay is mainly determined by the usage of Amazon, but not the latest app usage (WhatsApp). In this case, WhatsApp is a drop-in app that may confuse our app prediction model. The general RNN unit treats all the events in the sequence equally. To minimize the impact of drop-in apps, we incorporate an attention mechanism into our RNN-based app prediction framework. Our RNN model outputs a fused feature vector that is a weighted combination of all the hidden states. The weight of each hidden state represents the importance of each historical app usage event, which is learned by a soft attention mechanism [14].

The spatial app usage pattern is that app usage behavior is highly related to spatial context [43]. We add the spatial context into *ATPP* through a spatial context feature extractor. With our dataset, we know the location of the associated cell tower when an app usage event occurs. We leverage the set of **Points of Interest (POIs)** surrounding the cell tower to capture the spatial features of every app usage event. Based on such a representation, *ATPP* can generalize the past spatial app usage patterns to new places by comparing the similarity between POI vectors at different places.

To accelerate the learning process, we develop an efficient app representation module. The input to the above RNN-based app prediction is each app usage event in an app usage sequence. Normally, a one-hot vector is used to represent an app; i.e., all bits in the vector are "0" except one "1" to specify that app. The size of the one-hot vector is the number of apps installed by a user. However, such a one-hot vector cannot capture the similarity between different apps. The app usage behavior

Table 1. Notations Used in This Article

Notation	Description
$e = (t, a)$	A user opens app a at timestamp t
$\mathcal{H}$	Historical app usage data
$\mathbf{h}$	The hidden state
$\lambda^*(t)$	The conditional intensity function of t
$m^*(a)$	The probability distribution of app a
$\mathbf{W}_{KD}$	The weight matrix in app representation module
$\mathbf{W}_{EL}$	The weight matrix in feature fusion
$\mathbf{v}_h$	The temporal-app vector
$\mathbf{v}_l$	The spatial context vector
$\mathbf{V}_s$	Weight matrix for app prediction
$\mathbf{v}_t$	Weight vector for time prediction
N	The length of time window
D	The dimension size of the temporal-app vector

learned from one app cannot be generalized to other apps with similar behavior. *ATPP* adopts a low-dimensional representation model. We use a deep neural network to convert a one-hot vector into a more expressive vector with a lower dimension; i.e., every item in the representation vector is a floating number. As a result, similar apps can share similar representations and utilize the learned app usage behavior for app prediction.

We implement *ATPP* on TensorFlow [58], an open-source machine learning platform. We train a set of *ATPP* parameters, including the GRUs' parameters of the app predictor, the weight matrix of the app representation module, and the spatial context feature extractor. We perform end-to-end training of all these parameters by using the Adam algorithm, which calculates the gradient of a loss function and updates all learning parameters accordingly.

We conduct both trace-driven validation and field experiments. The trace-driven validation is on an anonymized app usage dataset from 443 users over 21 days. The results demonstrate that *ATPP* provides high accuracy up to 81.5% in app ID prediction and 1.45-minute app time prediction. The field experiments involve 22 volunteers who use our app pre-loading application on smartphones over 21 days. *ATPP* can reduce the app loading time by 78.1% on average. Compared with the state-of-the-art method (DeppAPP [42]), *ATPP* can reduce energy consumption by 8.9% and memory cost by 33.5% on smartphones.

In summary, this article makes the following contributions:

- To the best of our knowledge, we are the first to model the app prediction problem as an MTPP process for jointly predicting both the next app and its open time.
- We customize our MTPP-based framework by considering unique challenges in our app prediction system, including RNN-based MTPP model learning, app usage event presentation, and temporal dependency and spatial context feature extraction.
- We conduct extensive experiments on a large-scale app usage dataset and a field experiment with 22 users. The results demonstrate that *ATPP* outperforms state-of-the-art methods.

2 MTPP-BASED APP PREDICTION

In this section, we model app usage of a mobile user by MTPP and leverage the MTPP model to predict the next app and its open time. We also discuss the limitations of such a simple method and the challenges to developing a practical MTPP-based app prediction solution. Table 1 presents the notations used in this work.

2.1 Simple MTPP-based App Prediction Method

We model the app usage behavior of each user as an MTPP process [1] to capture the dynamics of his or her app usage behavior. MTPP is a random process generated by an ordered sequence of events in time:

$$\mathcal{H} = \{e_0 = (t_0, a_0), e_1 = (t_1, a_1), \ldots, e_N = (t_N, a_N)\}, \tag{1}$$

where a_i is the app opened by a user at time t_i. MTPP characterizes the app usage time by a conditional intensity function $\lambda^*(t)$, which is the probability of observing an event in a time window $[t, t + dt)$ given the historical events $\mathcal{H}$:

$$\lambda^*(t) := \mathbb{P}\{\text{apps are opened in } [t, t + dt) \mid \mathcal{H}\}, \tag{2}$$

where the sign $*$ means that the intensity function depends on the history $\mathcal{H}$. We can specify a probability that the app a_i will be used in the next time window $[t, t + dt)$ given $\mathcal{H}$:

$$m^*(a_i) := \mathbb{P}\{\text{app } a_i \text{ is used in } [t, t + dt) \mid \mathcal{H}\}. \tag{3}$$

Model Specification. To use the above MTPP model, we need to first specify the probability $\lambda^*(t)$ and $m^*(a_i)$. We will find a specific MTPP model that can mostly capture app usage behavior and then use historical data to determine the parameter of that MTPP model. In this work, we choose the Hawkes process [23] to model app usage behavior, which has been used in many applications to model time sequence data. In the Hawkes process, the intensity function $\lambda^*(t)$ is defined as follows:

$$\lambda^*(t) = \mu + \sum_{t_i < t} \kappa(t - t_i), \tag{4}$$

where μ is a baseline intensity independent of the historical data, and $\kappa(t)$ is a triggering function. A common choice of the triggering function $\kappa(t)$ is an exponential function:

$$\kappa(t) := \alpha\omega \exp(-\omega t), \tag{5}$$

where ω is used to control the rate of decaying influence from previous events, and α controls the likelihood of an event causing another event. Recent events will increase the value of the intensity function if $\kappa(t)$ is greater than 0.

At the same time, we assume app ID usage as a multinomial distribution [1], i.e., the probability that app a_i will be used in the next time window $[t, t + dt)$ is determined by its usage frequency in the historical events $\mathcal{H}$:

$$m^*(a_i) = \frac{\exp(f_{a_i})}{\sum_{a_i=1}^{a_K} \exp(f_{a_i})}, \tag{6}$$

where K is the number of apps installed on the smartphone, and f_{a_i} is the frequency that a_i is opened, corresponding to the ratio between the usage number of a_i to the total usage number of all apps.

Prediction of Next App and the Open Time. We can use Equation (6) to predict the next app that is the app with the highest usage frequency. At the same time, we can use Equation (7) to predict the open time of the next app usage [1]:

$$\hat{t_{i+1}} = \int_{t_i}^{T} t \cdot \lambda^*(t) \exp\left(-\int_{t_i}^{t} \lambda^*(\tau) d\tau\right) dt, \tag{7}$$

where $\lambda^*(t) \exp(-\int_{t_i}^{t} \lambda^*(\tau) d\tau)$ is the probability density function. It represents the likelihood that an app usage event will occur at the time t given the history. Using Equation (7), we can predict the open time of the next app by calculating the expectation of the next app usage event time. In our implementation, T is 1 hour. Hence, it will predict the user's app usage behavior in the next hour.

Table 2. Examples of App Usage Data

UserID	OpenTime	LAC	CID	AppName	Duration(s)
4C1F9	201804*11	42*3	31*9	App 1	10
1V6G3	201805*52	51*9	34*8	App 2	34
6P1H2	201805*56	43*8	25*7	App 1	26

Parameter Learning. In the above MTPP model, three parameters need to be learned, denoted as $\theta = (\mu, \alpha, \omega)$, which model different app usage behaviors of different users. We use **Maximum Likelihood Estimation (MLE)** to find the optimal values of these parameters for each user by maximizing the likelihood function. Given the app usage sequence $\mathcal{H}$ and intensity function, we can compute the likelihood function of $\mathcal{H}$ as

$$L = \left(\prod_{i=1}^{N} \lambda^* (t_i, a_i)\right) \exp\left(-\int_0^T \lambda^*(s)\mathrm{d}s\right), \tag{8}$$

where $\lambda^*(t, a_i)$ is the conditional intensity function for the usage event of app a_i in next time window $[t, t + dt)$. We can calculate $\lambda^*(t, a_i)$ based on Equations (4), (5), and (6). The likelihood function is the joint density function of all the app usage events in the observed data $\mathcal{H}$. The last term in Equation (8) represents the probability of no app used at $t \in [0, T]$ *except* $\{t_i\}$. With the likelihood in Equation (8), we can use our data points to find the best settings of the three parameters $\theta = (\mu, \alpha, \omega)$, by maximizing the likelihood.

2.2 Limitations and Challenges

We briefly introduce the app usage dataset used in this work. We conduct a set of experiments based on the dataset. According to the experiment results, we find that the above simple MTPP-based app prediction solution provides limited performance; i.e., the mean absolute error of open time prediction is 4.36 minutes. The details of the experiment settings will be introduced in Section 5.2. The low performance of the simple MTPP-based method is mainly caused by three challenges that are ignored in the solution.

Dataset. We use a large-scale dataset of app usage logs from 443 users collected over 21 days by a mobile operator. It contains 2,104,369 app usage records. Users send HTTP requests to the cellular tower by clicking on apps; the cellular tower then sends URLs to the operator server. They then parse URLs logs to the corresponding app ID by URL-app encoding tables maintained by the mobile operator. We merge the consecutive requests from the same app to get the app usage duration. Users can connect to the cellular tower at any location as soon as their smartphone is on. So, we know the user's approximate location area based on **Location Area Code (LAC)** and **Cell Tower ID (CID)**. It is used to identify a cellular tower uniquely. As shown in Table 2, each record consists of anonymized **user identification (UserID)**, open time, LAC, CID, app ID, and duration.

Challenge 1: Limitations of the above simple MTPP-based method. Figure 1 shows the probability of app usage of two users in our dataset during different hours of 1 day. The probability is calculated by the data of 21 days. We find that the probability of app usage is diverse at different hours, and the app usage patterns of two users are different over the same hour. However, the simple MTPP-based method makes strong assumptions about the generation process of app usage behavior, i.e., Hawkes process [23], which may or may not express the real app usage behavior. The conditional intensity function contains some parameters, which may restrict the expressive power of the app usage behavior. Therefore, it may be difficult for the Hawkes processes to fit the real app usage behavior. We need a more flexible conditional intensity function to capture the complex mechanisms behind app usage behavior.

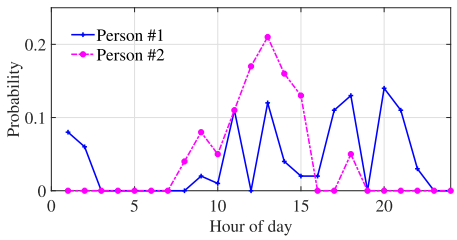


Fig. 1. The app usage probability that two subscribers open apps during different hours.

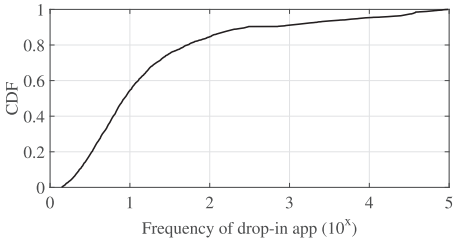


Fig. 2. The CDF of the total occurrence number of drop-in apps in 21 days for all users.

Challenge 2: Drop-in app usage. Drop-in app usage behavior refers to the tendency of users to open apps by chance while engaged in other activities such as watching videos or playing games. Figure 2 depicts the total occurrence number of drop-in apps that happened in our dataset. There is a 95.3% probability that drop-in apps will occur 10,000 times. This high occurrence rate may negatively impact the model's performance. Therefore, historical app usage events may have different influences on the current prediction due to drop-in apps. We propose an attention-based feature extractor to extract such influences. It can determine which app usage is most important for the current prediction.

Challenge 3: Spatial-related app usage. As found in many existing app prediction works [43, 50], environment context (e.g., spatial context) has a significant influence on the app usage behavior. The challenge is to incorporate spatial information into our MTPP-based app prediction framework that can tackle this challenge and the above two challenges in a unified framework. We will build a spatial context feature extractor in our framework to characterize the spatial context for app prediction.

3 DESIGN OF *ATPP*

In this section, we introduce the design of our system and its key components to deal with the above three challenges.

3.1 Architecture of *ATPP*

Figure 3 depicts the architecture of *ATPP*, consisting of three key modules, i.e., an app representation module, an app usage event predictor, and a context-aware optimization module. An app usage sequence is a sequence of app usage events. Each app usage event is recorded as an app ID with a unique timestamp. Given an app usage sequence, we first use an app representation module to convert it into a sequence of high-quality representations for each app usage event (Section 3.2). Then, an RNN-based app usage event predictor is applied to predict the app ID that the user most likely uses and its open time (Section 3.3). At the same time, we customize the predictor model by incorporating a context-aware optimization module to improve the prediction accuracy, including an attention-based temporal feature extractor and a spatial feature extractor (Section 3.4).

The above prediction (inference) process includes three neural networks, including an app representation module (a two-layer neural network), an RNN-based app predictor, and a temporal feature extractor (a neural network of three fully connected layers). We perform end-to-end training for all these neural networks at the same time (Section 3.5). We design a loss function to quantify the quality of each app prediction result, which combines the result of both app ID prediction and app open time prediction. Based on the loss function, the Adam algorithm with descending gradient optimization will be used to update the weight parameters in all these neural networks simultaneously.

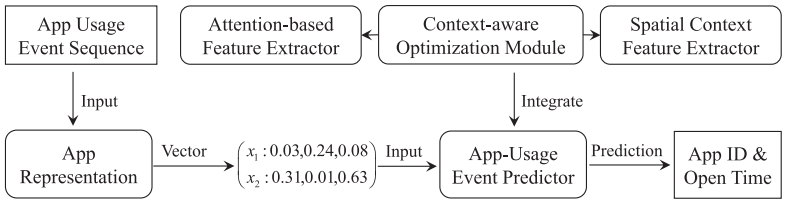


Fig. 3. The architecture of *ATPP*. The app representation module converts an app usage sequence into a sequence of representation vectors. The app usage predictor is an RNN-based model, integrated with two feature extractors.

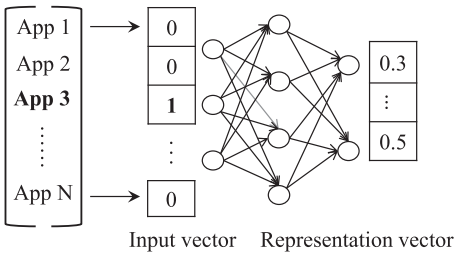


Fig. 4. The app representation module.

3.2 App Representation

An app in an app usage sequence can be intuitively represented by a one-hot vector, in which all bits are "0" except one "1." For example, in Figure 4, to represent App 3, only the third item in the one-hot vector is set to "1." Such a simple representation method suffers from two drawbacks. First, it is hard to train the app usage event predictor model that takes app representation as input, because the one-hot vector is too sparse, with too many "0s" especially when a user installs a large number of apps on his or her smartphone. Second, such a simple app representation cannot capture the similarity of apps. As a consequence, we cannot utilize the learned app usage behavior to infer some unobserved apps. Therefore, we develop an app representation module to automatically learn a low-dimensional representation for each app usage event.

Figure 4 depicts the architecture of the app representation module that uses a two-layer neural network to convert a high-dimension one-hot vector into a low-dimension representation vector. The input vector is a K-dimension one-hot vector, where K is the number of apps installed on a user's smartphone. The input one-hot vector is transformed into a D-dimension vector by a two-layer neural network with a weight matrix $\mathbf{W}_{KD}$.

For each app usage event, besides the app ID representation, the app usage event predictor also needs the time information of that event. We take the inter-event duration as the time information. Specifically, the inter-event duration between the ith app usage event and the previous app usage event is calculated as $t_i - t_{i-1}$, where t_i is the open time of app a_i in the sequence $(t_i, a_i)_{i=1}^{N}$. In our implementation, the inter-event duration is quantified in minutes. By obtaining the app representation and time information of each app usage event, we feed them into an app usage event predictor.

3.3 App Usage Event Predictor

We design a recurrent MTPP-based app prediction system, which leverages an RNN model to learn the intensity function of the MTPP process in a data-driven manner. Figure 5 shows the architecture of our app usage event predictor. A recurrent neural network composed of GRUs is

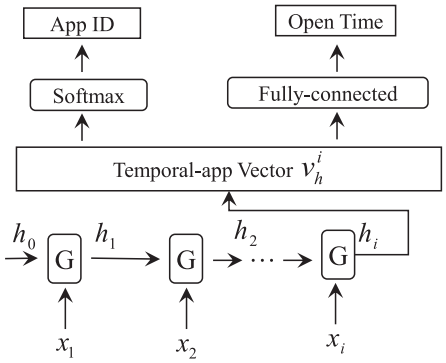


Fig. 5. The architecture of the app usage event predictor.

used to encode N app usage events into an app usage feature vector $\mathbf{v}_h^i$, which will be further passed to a *softmax* layer for predicting the next app ID and to a fully connected layer for predicting open time. N is the length of the input app sequence. In our current implementation, N is set to 5. The input of the app usage event predictor is a sequence of apps and their open time $\{\mathbf{x}_i\}_{i=1}^N$.

RNN-based feature extraction. We implement RNN as a set of GRU units [7]. Although **Long Short-Term Memory (LSTM)** [21] is also widely used, GRU achieves similar performance in many tasks with less computation [7]. The computation of GRU can be expressed as follows:

$$\mathbf{h}_i = \text{GRU}\left(x_i, \mathbf{h}_{i-1}\right), \tag{9}$$

where $\mathbf{h_i}$ is the hidden state, and $\mathbf{h}_{i-1}$ is the previous hidden state. The current hidden state, h_i, learns a feature representation that characterizes the dependency over previous app usage events.

At the beginning of the training process, h_0 is uniformly initialized to random values $[-0.1, 0.1]$ for the first app usage sequence. To be consistent, in the rest of the article, the current hidden state h_i is also referred to as the temporal app vector $\mathbf{v}_h^i$, i.e., $\mathbf{v}_h^i = h_i$. Based on the temporal app vector, we predict the next app $\hat{a_{i+1}}$ and open time $\hat{t_{i+1}}$.

Prediction of next app ID. We use a *softmax* layer to process the temporal app vector $\mathbf{v}_h^i$. The optimal predicted app $\hat{a_{i+1}}$ is calculated by selecting the corresponding maximum probability:

$$\hat{a_{i+1}} = \text{argmax}\left\{softmax\left(\mathbf{V}_s \cdot \mathbf{v}_h^i\right)\right\}, \tag{10}$$

where $\mathbf{V}_s$ is a $K \times D$ matrix that needs to be learned, and D is the dimension of vector $\mathbf{v}_h^i$. The term $\mathbf{V}_s \cdot \mathbf{v}_h^i$ will generate a K-dimension vector. The *softmax* layer normalizes the elements of the K-dimension vector into a probability distribution over K apps; i.e., each element is between 0 and 1, and the sum of all elements is 1.

Prediction of next app open time. We first need to learn an MTPP conditional intensity function via the RNN output $\mathbf{v}_h^i$. Inspired by RMTPP [13], we use the learned hidden states to calculate a general representation of the intensity function. Compared to the simple MTPP-based app prediction method that uses a predefined intensity function (e.g., Equation (4)), Equation (11) provides a general representation of intensity function that can be learned from historical data:

$$\lambda^*\left(t\right) = \exp\left(\mathbf{v}_t^\top \cdot \mathbf{v}_h^i + w_t \cdot \left(t - t_i\right) + b_t\right), \tag{11}$$

where $\mathbf{v}_t$, w_t, and b_t are parameters to learn, and t_i is the open time of app a_i. $\mathbf{v}_t$ is a parameter vector and its dimension is the same as $\mathbf{v}_h^i$. In Equation (11), the first part, $\mathbf{v}_t^\top \cdot \mathbf{v}_h^i$, characterizes the accumulative influence from previous app usage events on the open time of the next app. The second part, $w_t \cdot (t - t_i)$, emphasizes the influence of the latest app usage event (t_i, a_i). The last term,

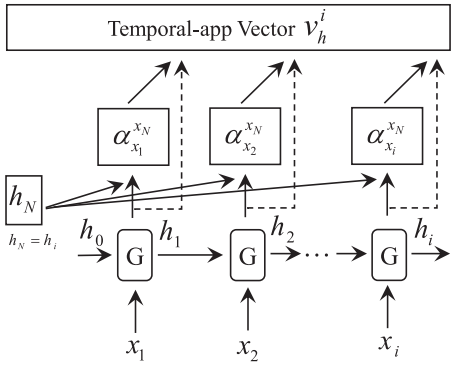


Fig. 6. The architecture of attention-based feature extractor.

b_t, represents a base intensity level. The exponential function outside provides a non-linear transformation that makes the intensity function positive. Based on the conditional intensity function obtained from Equation (11), we can predict the next app open time $\hat{t_{i+1}}$ through Equation (7).

3.4 Context-aware Optimization Module

To further improve the prediction accuracy of *ATPP*, we exploit two special app usage behaviors observed from our dataset, i.e., drop-in app usage and spatial-related app usage. To incorporate these behaviors into our app usage event predictor, we design two feature extractors, i.e., an attention-based feature extractor and a spatial context feature extractor. We update the temporal app vector $\mathbf{v}_h^i$ by fusing the above two features.

3.4.1 Attention-based Temporal Feature Extractor. In the previous app usage event predictor, we only use the current hidden state $\mathbf{v}_h^i$ to do prediction, but $\mathbf{v}_h^i$ does not consider the different contributions that previous app usage events may make to the next app prediction, especially when drop-in apps have been opened in the app sequence.

Figure 6 depicts the architecture of the attention-based feature extractor, which integrates a soft attention mechanism into *ATPP* to handle drop-in app usage, which calculates a vector as a weighted sum of a set of hidden states. Instead of only using the last hidden state $\mathbf{h}_i$ for app prediction, we use all the hidden state vectors $\{\mathbf{h}_i\}_{i=1}^N$ to generate a fused vector in this section. We can obtain a new temporal app vector $\mathbf{v}_h^i$ as follows:

$$\mathbf{v}_h^i = \sum_{i=1}^{N} \alpha_{x_i} \mathbf{h}_i. \tag{12}$$

The new temporal app vector $\mathbf{v}_h^i$ is a weighted sum of all hidden state vectors $\{\mathbf{h}_i\}_{i=1}^N$. The weight α_{x_i} is calculated by

$$\alpha_{x_i} = |\tanh(\mathbf{h}_i * \mathbf{h}_N)|, \tag{13}$$

where $\mathbf{h}_N$ is the latest hidden state in a sequence, and tanh is the score function measuring the influence strength from $\mathbf{h}_i$ to $\mathbf{h}_N$. If the hidden state $\mathbf{h}_i$ is similar to $\mathbf{h}_N$, the score function tanh generates a high weight, otherwise a low weight. More attention should be paid to the app usage event with a higher weight. We normalize all the weights to make sure that they are summed to 1.

3.4.2 Spatial Context Feature Extractor. A user may have diverse app usage behavior in different spatial contexts [43]. Hence, we develop a spatial context feature extractor to incorporate the

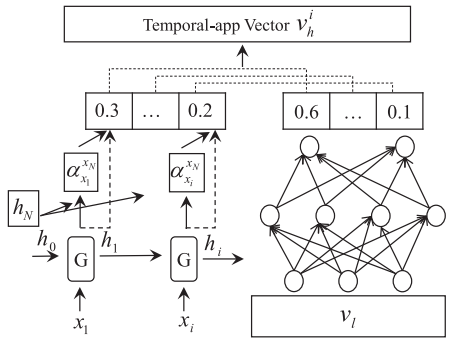


Fig. 7. Combination of the attention-based temporal feature extractor and spatial feature extractor.

spatial context into our app predictor. Figure 7 depicts the architecture of the spatial context feature extractor. It first generates spatial context based on the cellular data we used in this work. It then fuses the learned spatial context feature into the temporal app vector $\mathbf{v}_h^i$ learned above.

In our cellular dataset, the GPS coordinates are of the cell tower a user's mobile phone is associated with when the user's current app sends a request to the cell tower. GPS location (latitude and longitude) of the cell tower restricts the representation of spatial information for a user. To this end, we leverage the POI distributions nearby the location of a cell tower to represent the spatial context of a user. POI refers to all geographical objects that can be abstracted as points, such as restaurants and supermarkets.

In particular, for a sequence of N app usage events, we take the average values of N longitudes and latitudes as the central GPS coordinates. We then characterize the central GPS coordinates by the distribution of POIs within a radius of 500 meters. We obtained a POI dataset containing more than 300,000 POIs of the city from AMAP [37], which provides **application programming interfaces (APIs)** to crawl POIs on the map. For a specific location, we represent the 23-dimension spatial vector as $\mathbf{v}_l = [l^1, l^j, \ldots, l^m]$ for 23 types of POIs. The dimension corresponds to the number of categories of POIs, where l^j is the POI number of type j within the radius of 500 meters.

3.4.3 Feature Fusion. We apply two methods to combine the spatial context feature vector $\mathbf{v}_d$ and the temporal app vector $\mathbf{v}_h^i$ obtained from the attention-based feature extractor. A simple way is to concatenate these two feature vectors into a long feature vector. Another method is to use the Hadamard product to perform the element-wise multiplication of these two feature vectors.

The Hadamard product requires two feature vectors that should have the same dimension. Therefore, we input the spatial context feature vector $\mathbf{v}_l$ into a neural network, which is composed of three fully connected layers. It can be specified by its parameters $\mathbf{W}_{EL}$. After this transform, the size of the spatial context feature vector becomes the same as the temporal app vector $\mathbf{v}_h^i$ obtained from the attention-based feature extractor module.

Based on our experimental results (see Figure 10 in Section 5.2.2), we can find that the Hadamard product provides better performance in the app prediction. Using the Hadamard product, we obtain a new temporal app vector $\mathbf{v}_h^i$. We can use the new generated vector to predict the next app and its open time, as introduced in Section 3.3.

3.5 End-to-end Training

In *ATPP*, there are a set of neural network parameters to learn, including the matrix $\mathbf{W}_{KD}$ of the app representation module, the GRU's parameters of the app predictor, and the matrix $\mathbf{W}_{EL}$ of the spatial context feature extractor. We maximize the log-likelihood ($\mathcal{LL}$) [13] of observing app usage

event sequences $\mathcal{H} = \{t_i, a_i, l_i\}_{i=1}^{N}$ to train these parameters:

$$\mathcal{LL}(\mathcal{H}) = \sum_{i=1}^{N} a_i \cdot \log\left(\mathbf{u}_{a_i} | \mathcal{H}(t_i, a_i, l_i)\right) + \log\left(f(t_i | \mathcal{H}(t_i, a_i, l_i))\right), \tag{14}$$

where $\mathbf{u}_{a_i} = softmax(\mathbf{V}_s \mathbf{v}_h^i)$ is the probability that all apps are used. The first term is the probability that app a_i is used, and the second term is the probability of outputting the true value t_i, which assumes that the error between the open time prediction and the true value obeys the Gaussian distribution:

$$f(t_i | \mathcal{H}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\left(t_i - \hat{t}_i\right)^2}{2\sigma^2}\right), \tag{15}$$

where σ is the standard deviation and its default value is set to 1. The estimated $\hat{t}_i$ was obtained from Equation (7). To maximize log-likelihood, we can minimize the loss function of app ID and open time prediction with the Adam algorithm, which can be obtained from Equation (14). In this way, we can train the *ATPP* model in an end-to-end manner.

4 IMPLEMENTATION

In this section, we introduce the implementation details of *ATPP*, which contains three modules. We also implement an Android application that is used to run *ATPP* on commodity smartphones.

4.1 Model Implementation

We implement *ATPP* on the TensorFlow [58] platform. Three models in *ATPP* are implemented, i.e., an app representation module (Section 3.2), an app usage event predictor (Section 3.3), and a context-aware optimization module (Section 3.4).

First, we implement the app representation module as a two-layer neural network, which has K and D neurons, respectively. K is the dimension of the input one-hot app representation vector, and D is the dimension of temporal app vector $\mathbf{v}_h^i$. D is set to 64 based on the experiments of Section 5.2.4. This leads to the total number of parameters of the module $K * D$.

Second, for the app usage event predictor, the length (N) of the app usage sequence that is used to train the model is 5. We also use a two-layer fully connected feedforward neural network as the fully connected layer that is used to feed to the softmax layer. Specifically, the layer has D and K neurons in the two layers, respectively. We adopt the *relu* activation function for the first layer and the *tanh* activation function for the second layer. Moreover, to alleviate the over-fitting problem when training our system, we introduce an L_2 regularization term [36] in the loss function. This leads to the total number of parameters of the predictor $K * D + (N * N + N * D + N) * 4$.

Third, for context-aware optimization, we use a three-layer fully connected feedforward neural network that has $L(23)$, $D/2$, and D neurons in the three layers, respectively. Their activation functions are *relu* function, where 23 is the number of categories of POI provided by AMap [37]. This leads to the total number of parameters of our model $L * D/2 * D$. Furthermore, we conduct a grid search strategy to obtain the optimal settings of hyper-parameters in *ATPP*, as shown in Table 3.

In summary, to perform one inference, the number of parameters for our system is $K * D + K * D + (N * N + N * D + N) * 4 + L * D/2 * D$, corresponding to the space complexity $O(n^2)$. Similarly, the time complexity is $O(n)$.

Finally, when training the model, for simple MTPP-based app prediction methods, we use the first 14 days' data of each user to obtain their model's parameters. For the RNN-based predictor, we first use all users' 14 days of data to train a general model, then use 14 days of data from each

Table 3. Hyper-parameter Settings

Hyper-parameter	Setting
Batch size	32
Learning rate	0.001
Momentum	0.9
Decay steps	100
Decay rate	0.0001
L2 penalty	0.001
Standard deviation of Gaussian penalty σ	1.0

user to train his or her own RNN model. Training a good deep learning model usually requires a lot of data. And users might share similar app usage behaviors and exhibit some specific app usage patterns. We first train a general model on all users' data. Not only does it learn app usage behavior for all users, but also it ensures that the general model can be trained well on such a large dataset. We then train a personalized model for each user with their data based on the general model. The personalized model is trained by fine-tuning the general model based on each user's data. In this way, the personalized model not only had the knowledge of all users but also learned each user's unique app usage patterns.

We only use mobile CPUs for inference on smartphones, because mobile GPUs provide just as much performance as mobile CPUs on most Android devices, and mobile CPUs are still the most used because of their general availability, mature programming environment, and so on.

4.2 Application Implementation

ATPP is implemented as a customized application on smartphones (running on Android OS). The implementation of the app includes three modules, i.e., a context-sensing module, a background scheduler, and a model loader.

Context-sensing module. This module is used to obtain current app usage profiles of users, including currently used app, open time, and spatial context. Note that all sensitive information is acquired voluntarily. Specifically, the module obtains the currently used app through the *AccessibilityEventEvents* method provided by the Android SDK platform, obtains the status of the smartphone through the Android *logcat*, and obtains the location information through the *LocationManager*.

We obtain the user's location through *LocationManager* and take the cellular tower closest to the current location as the one linked by the smartphone of users. Moreover, we will pack the spatial feature of all cellular towers in the application. By doing so, we do not need additional data or any support from mobile carriers, and there is no network overhead when running *ATPP* on smartphones.

Background scheduler. We will pre-load the app when the time is close to the predicted app open time, which minimizes the overhead produced by pre-loading. We develop a background scheduler to pre-load the apps into memory before the apps' open time. We use *getLaunchIntent-ForPackage* in Android *PackageManager* to realize the pre-loading.

Model loader. We leverage mobile operator data to train the *ATPP* offline. After getting the well-trained model, we leverage TensorFlow Lite [34], a tool to run TensorFlow models on mobile devices that is widely used [22, 56], to integrate the trained model and data into the application. To make inference on the smartphone, we export the well-trained model to the *tf.GraphDef* file. After a user opens an app, our implemented app will make an inference on the smartphone to predict the next app ID and its open time.

Since TensorFlow Lite currently does not support training operations on mobile devices, we only run our app for inference without updating models on smartphones. In the future, given that we cannot train models on smartphones, we cannot design federated learning-based methods. Instead, we could design a centralized model where users transmit their data to the server, and the server trains a model for all users. We perform inferences on the server. The server then sends results to the smartphone. In this way, it reduces the training complexity. But it may raise privacy concerns for users and add to the communication overhead. To solve these two problems, we would utilize some encoding and decoding algorithms to protect users' privacy and reduce the communication overhead.

5 EVALUATION

We conduct a variety of experiments to evaluate our system through trace-driven evaluation and a field study of 22 volunteers.

5.1 Experiment Settings

Performance criteria. To evaluate our system, we use two metrics, i.e., Hitrate@K and **Mean Absolute Error (MAE)**. Hitrate@K calculates the hit ratio of the top-K apps, which is used to evaluate the accuracy of the next app prediction:

$$Hitrate@K = \frac{1}{|\mathcal{U}|}\sum_{u=1}^{|\mathcal{U}|}\frac{S_u(K)\cap T_u}{|T_u|}, \tag{16}$$

where $|\mathcal{U}|$ is the set of users, T_u is the test data of users u, and $S_u(K)$ is a set of top-K predicted next apps. Users can only open one app each time. When a user clicks any one of the top-K predicted apps, we consider it as one hit.

We adopt MAE to evaluate the accuracy of app open time prediction. MAE measures the absolute difference between the predicted timestamp and the ground truth:

$$MAE = \frac{1}{|\mathcal{U}|}\sum_{u=1}^{|\mathcal{U}|}\frac{1}{m}\sum_{i=1}^{m}\left|\hat{t}_i - t_i\right|, \tag{17}$$

where $\hat{t}_i$ and t_i are the predicted open time and the real value, respectively, its unit is minutes. When calculating the MAE, we ignore whether the predicted next apps are correct. The smaller the value of MAE, the better the performance on the predicting time. Then *ATPP* can save more energy consumed by pre-loading app into memory, which may improve the user experience of using smartphones.

Benchmarks. We compare the performance of *ATPP* with two types of existing solutions. First, DeepAPP [43] and AU2V [60] are based on deep learning models [51] that predict the next app. Second, APPM [38] performs best among the traditional methods, i.e., Markov or Bayesian models. Finally, we also develop three versions of the MTPP-based method to predict the next app and open time, including the **Homogeneous Poisson Process (HPP)**-based method, **Hawkes Process (HP)**-based method (HP), app usage event predictor (RP). There are two methods that predict app and time simultaneously, i.e., APPM and RP. DeepAPP predicts the apps that a user will open in the next time slot. AU2V only predicts the next app. HPP and HP are used to predict the open time of the next app.

- *DeepAPP [43].* It predicts the apps that a user will open in the next time slot based on reinforcement learning [9, 15, 32], which considers two pieces of context information, including the last used app and spatial feature. We set the time slot as 1 minute in our implementation.

- *AU2V [60].* It incorporates the app sequence, user personalized characteristics, and discrete temporal context to predict the next app that the user most likely opens.
- *APPM [38].* It uses the app sequence to compute the probability of the following app based on **Prediction by Partial Match (PPM)** model [8]. Moreover, it predicts app open time through a **Time Till Usage (TTU)** model.
- *HPP [1].* We model app usage behavior as HPPs. Its intensity function is constant. It is the simplest MTPP-based method to model the app usage behavior.
- *HP [23].* It assumes that app usage behavior is the Hawkes processes, which are introduced in Section 2.
- *RMTPP [13].* It uses a recurrent neural network to learn a representation of conditional intensity function from the event history. For convenience, we abbreviate it as RP.

Evaluation setup. We compare *ATPP* with the above baselines in Section 5.2.1, then evaluate the effectiveness of the Hadamard product in Section 5.2.2. We also discuss the effectiveness of context-aware optimization in Section 5.2.3. The default value of parameters is further discussed in Section 5.2.4. We measure the performance of ATPP under different scenarios, i.e., the number of dominant apps, installed apps, and app usage records, in Section 5.2.5. Next, we do field experiments with 22 users in Section 5.3, including the accuracy of each user and latency improvement. Finally, we compare the overhead produced by *ATPP* and DeepAPP in Section 5.4.

5.2 Trace-driven Evaluation

We conduct trace-driven evaluations on a large-scale app usage dataset introduced in Section 2.2. We divide the dataset into two parts, i.e., the 14-day dataset for training and the 7-day dataset for testing.

The dataset used in the trace-driven evaluations has an inherent limitation, which does not acquire the activities from apps that do not make HTTP requests, request by HTTPs, or access the internet through WiFi. However, we can collect all apps' activities by the app implemented in Section 4.2 during field experiments. It can log all apps' activities installed on the smartphone. Moreover, *ATPP* gives a similar performance in the field study, compared with trace-driven evaluation.

Privacy issues. To avoid user privacy disclosure, the operator replaced user identification with a hash code. The app usage data only contains anonymized records, as shown in Table 2, without any information relating to text messages or phone conversations. Besides, we randomly select from a very large dataset for our dataset, which can also prevent leaking the mobile users' privacy. In the field experiments, we also anonymize the user identifier by a hash code. We do not collect the precise location of the user for our system only needs the POI distribution information around cell towers. *ATPP*'s code and some desensitized data are released on the website.[1]

Parameter settings. We set the default value of the size of the time window N to 5 and the dimension of the temporal app vector D to 64. We use these settings by default to conduct the following experiments. In Section 5.2.4, we will explain how we set them to the optimal values.

5.2.1 Overall Performance Comparison. As shown in Figure 8, *ATPP* performs best in Hitrate@K. Compared with other models, there is an improvement in our system. Concerning Hitrate@5, *ATPP* achieves a substantial improvement, making it around 8.8% higher than APPM, 6.1% higher than RP, 6.0% higher than DeepAPP, and 3.3% higher than AU2V. The reason for this improvement is that the attention-based feature extractor module captures the weighted influence of historical app usage event sequences for the prediction task. The spatial context feature extractor module denotes personalized app usage patterns in specific locations.

[1] https://sites.ucmerced.edu/wdu.

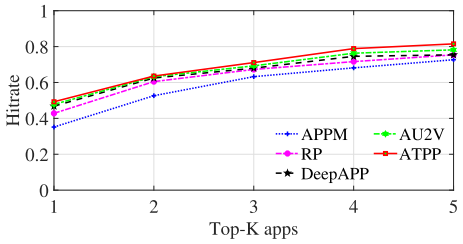


Fig. 8. Performance criteria: Hitrate.

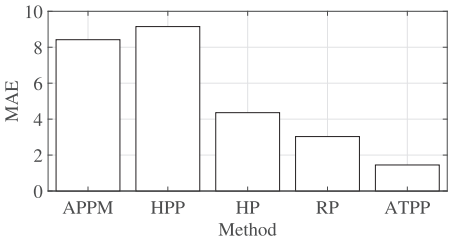


Fig. 9. Performance criteria: MAE.

APPM gives the worst performance, for it just utilizes the app sequence information through the PPM [8] model. It doesn't take context information into account. RP achieves better performance than APPM (75.4% vs. 72.7%) in Hitrate@5, for it leverages a longer app usage sequence through a recurrent neural network. DeepAPP gives a little worse performance than *ATPP* (75.4% vs. 81.5%) in Hitrate@5. DeepAPP is based on deep reinforcement learning. It models the app usage behavior as a one-order Markov Decision Process, which can only consider the influence of the last app users used recently. AU2V mainly leverages app sequences to make predictions through attention mechanism and incorporates user ID and temporal context, which achieves 78.2% in Hitrate@5. However, it cannot be used efficiently in practice for it does not support predicting open time.

Although *ATPP* provides a marginal improvement on the prediction of the next app, *ATPP* can accurately predict app open time. As shown in Figure 9, *ATPP* performs best in MAE. *ATPP*achieves a significant improvement in MAE, giving it around a 6.31× reduction compared to HPP, 3.01× reduction compared to HP, 5.80× reduction compared to APPM, and 2.09× reduction compared to RP. The reason for making such an improvement is that simple MTPP-based models make strong assumptions about app usage behaviors. RP leverages a recurrent neural network to model intensity function, but it doesn't consider drop-in app usage and spatial-related behavior.

To sum up, in terms of app and time prediction, *ATPP* outperforms APPM by 8.8% and 5.80×, respectively. And our system performs better than RP by 6.1% and a 2.09× reduction, respectively. Thus, *ATPP* can provide significant accuracy gains for both the next app and time predictions.

5.2.2 Performance Gain of Hadamard Product. There are two methods to combine the attention-based feature extractor and the spatial context feature extractor, i.e., concatenation or Hadamard product. Figure 10 presents the performance of the two methods. "ATPP-C" denotes the operation of concatenation. The Hadamard product gives better performance than concatenation. Hence, we choose the Hadamard product to integrate two feature extractors.

5.2.3 Effectiveness of Two Modules in Context-aware Optimization. We investigate the improvement in the performance of the two feature extractors from the context-aware optimization module, i.e., the attention-based temporal feature extractor and the spatial context feature extractor, denoted as *AF* and *SF*, respectively. We take RP as the baseline and use Hitrate@5 and MAE to evaluate the effectiveness of the two modules, as shown in Table 4. "RP+AF" indicates the app usage event predictor with the module of *AF*. "RP+SF" means predictor with the module of *SF*, which directly integrates the last hidden state with spatial vector through the Hadamard product. "HGain" is the gain of Hitrate@5, and "MGain" is the gain of MAE.

ATPP provides a 6.1% performance gain in Hitrate@5 and 2.29× reduction in MAE for it efficiently fuses drop-in app usage and spatial-related app usage by a context-aware optimization module.

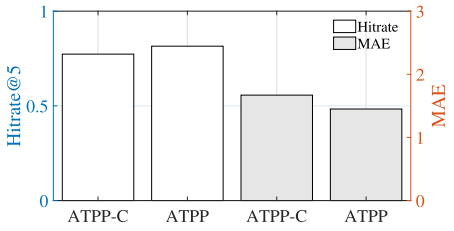


Fig. 10. The gain of Hadamard product.

Table 4. Effectiveness of Proposed Modules

Model	Hitrate5	HGain	MAE	MGain
RP	75.4%	–	3.03	–
RP+AF	80.2%	4.8%	1.69	1.79×
RP+SF	76.5%	1.1%	2.95	1.03×
ATPP	81.5%	6.1%	1.45	2.09×

Attention-based temporal feature extractor. This component improves the prediction performance by around 4.8% higher than the baseline in Hitrate@5 and a 1.79× reduction in MAE. The module incorporates drop-in app usage into the system, which gives significant performance improvement in predicting the next app usage. It shows the substantial impact of the app attention mechanism on app usage prediction, because it can capture the user's intention in using apps and detect which app is important to predict the next app. The model can perform well in scenarios when there are some drop-in apps, which are interrupting the app usage patterns.

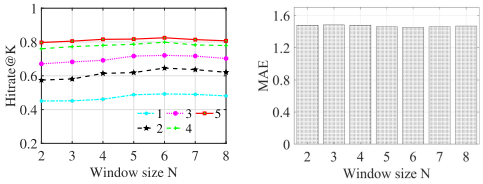
Spatial context feature extractor. Compared to the baseline model, considering spatial context obtains a 1.1% performance gain in Hitrate@5 and 1.03× reduction in MAE. The benefit comes from people's tendency to use different apps in different environmental contexts. For example, at home, people are more likely to use games or videos apps. Social apps are used more frequently at shopping malls. We use the POI distribution near the app usage location to represent the environmental context. However, with the development of commercialization, the POI distribution could be similar among different locations in a city. At home or a company, their surroundings have similar POI distribution. It might result in our model not learning spatial app usage patterns efficiently.

5.2.4 Parameter Settings. We further test the choice of two parameters in *ATPP*, i.e., the length of app usage event sequences N and the dimension of temporal app vectors D.

Window size. We first investigate the performance of *ATPP* with varying window size N (length of app usage event sequences). As shown in Figure 11, we test Hitrate@K (K = 1, 2, 3, 4, 5) by varying the window size from 2 to 8. It can be seen that Hitrate@1 increases when the window size varies from 2 to 5. It suggests that adding the latest app usage event sequences provides more historical information for the prediction task. However, some results slightly decrease when the window size varies from 5 to 8. That is because inputting such a long app usage event sequence increases the difficulty in training our system [17]. Moreover, if we input longer sequences to predict the next app, there are relatively little training data. Given that the transformer model stacks several attention blocks to learn long-term temporal information [47], we may replace the RNNs with transformers in the future. In this way, *ATPP* could effectively learn temporal features from long app usage sequences with sufficient training data.

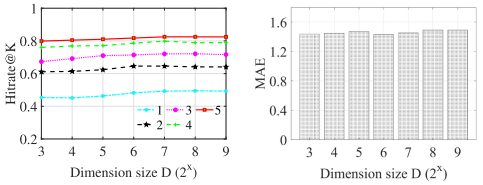
This trend of change in MAE is similar to Hitrate@K. We can see that the value of MAE decreases with a window size from 2 to 5. The system achieves the best Hitrate@K and MAE metric when the window size N is 5.

Dimension size. We test the performance of *ATPP* with varying temporal app vectors' dimension D when the window size is 5, shown in Figure 12. Hitrate@1 increases quickly when the size varies from 2^4 to 2^7 and increases slowly from 2^3 to 2^4. The system achieves the best Hitrate@K when the dimension size is 2^6. The MAE increases when the dimension size varies from 2^6 to 2^9, and the MAE is best when the dimension size is 2^6.



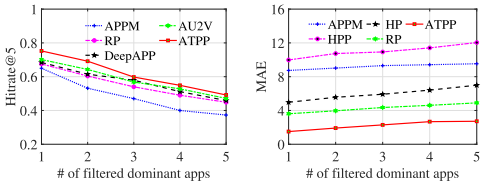
(a) Hitrate@K (b) MAE

Fig. 11. Effect of the window size N.



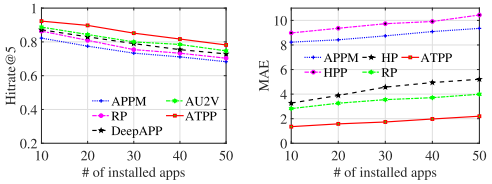
(a) Hitrate@K (b) MAE

Fig. 12. Effect of the dimension of vector D.



(a) Hitrate@5 (b) MAE

Fig. 13. Effect of dominant apps.



(a) Hitrate@5 (b) MAE

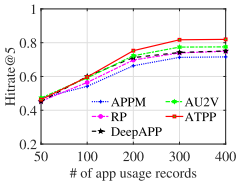
Fig. 14. Effect of the number of installed apps.

The ratio of the training data to the testing data. Table 5 shows the variation of accuracy as the ratio of the training data to the testing data varies. We discover that the increase in ratio improves the accuracy but shows a downward trend when the ratio is larger than 14:7. This is because more training data will learn more app usage patterns but may cause over-fitting of our predictive model. We know that with less training data, the model's parameter estimates have greater variance. With less testing data, our accuracy statistics will have greater variance. We should focus on partitioning the data so that none of the variances are too high. They are two competing concerns. If the training data is too small, the model might have an over-fitting problem, which can lead to low accuracy on a large test dataset. On the other hand, if the size of the test data is too small, it can lead to a high variance in the results. While this high variance might be reduced by k-fold cross-validation, there is still a marginal reduction in the performance. We can see from Table 5 that although the ratio of 14:7 provides the highest performance, the ratios of 15:6 and 18:3 are just a little worse than the ratio of 14:7. But if the training data is small, such as 9:12 or 11:10, they have large differences with the best performance. It means that there is not enough data to well-train models. Therefore, we select the ratio at 14:7 with the best performance.

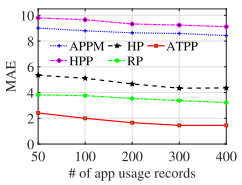
5.2.5 Performance under Different Scenarios. The above experiments prove the effectiveness of *ATPP* in our dataset. We further study the effects of different scenarios on our system performance, such as removing dominant apps, varying the number of installed apps, and app usage records.

Number of dominant apps. We investigate the influence of taking out dominant apps (the most frequently used apps) on our system. We change the number of dominant apps removed, then evaluate the prediction performance. As shown in Figure 13, the performance of *ATPP* is the best among all methods. It gives the Hitrate@5 of 75.1%, compared with 64.9% in APPM, 67.5% in RP, 68.4% in DeepAPP, and 70.1% in AU2V. It also performs best in MAE, giving an MAE of 1.52, compared with 9.98 in HPP, 4.97 in HP, 1.11 in RP, 8.74 in APPM, and 3.61 in RP. The results verify that our system can perform well in predicting the apps that are not used frequently.

Number of installed apps. If numerous apps are installed on users' smartphones, the prediction task becomes more difficult. This experiment explores how *ATPP* performs when the number of apps (M) installed on smartphones changes. We classify the number of installed apps into five

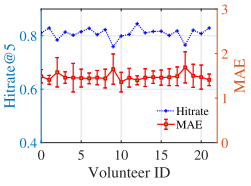


(a) Hitrate@5

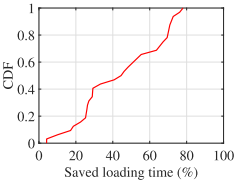


(b) MAE

Fig. 15. Effect of number of app usage records.



(a) Accuracy



(b) Latency improvement

Fig. 16. Accuracy and latency analysis.

Table 5. Ratio of the Training Data to the Testing Data

Ratio	9:12	11:10	14:7	15:6	18:3
Hitrate@5	70.4%	76.9%	81.5%	80.7%	80.2%
MAE	1.93	1.71	1.45	1.56	1.59

levels, including $\{M < 10\}$, $\{M >= 10 \ \& \ N < 20\}$, $\{M >= 20 \ \& \ M < 30\}$, $\{M >= 30 \ \& \ M < 40\}$, and $\{M >= 40 \ \& \ M < 50\}$. As shown in Figure 14, the Hitrate@5 decreases, and MAE increases as the number of installed apps increases. The experiment results demonstrate that the fewer installed apps there are on smartphones, the easier it is for *ATPP* to predict next app.

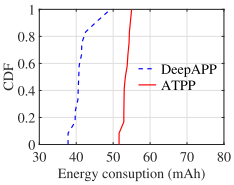
Number of app usage records. The number of app usage records also influence the performance of our system. We study how *ATPP* performs when the number of records (H) changes. We also classify it into five levels, i.e., $\{H < 50\}$, $\{H >= 50 \ \& \ H < 100\}$, $\{H >= 100 \ \& \ H < 200\}$, $\{H >= 200 \ \& \ H < 300\}$, and $\{H >= 300 \ \& \ H < 400\}$. Figure 15 demonstrates the performance of the different number of usage records. As the number of records increases, the Hitrate@5 improves and MAE decreases. It suggests that we can train the system better if there is a larger number of records.

5.3 Field Study

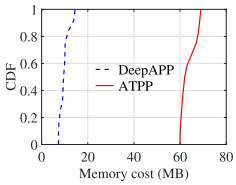
We test *ATPP* over 21 days. We deploy the system as the architecture in Figure 3. We recruit 22 volunteers. They include 7 females and 15 males, aged from 17 to 48, who have different occupations, e.g., company employees, college teachers, and students. After volunteers agree to the experiment, we install the Android app on their smartphones. First, we let users install our Android app. The app collects 14 days of data used to train a general model. We also train each user's model on their own dataset based on the general model. After that, we load the trained model into our app by TensorFlow Lite. Specifically, we export DNN models to a *tf.GraphDef* file through the interface (*tensorflow.gfile.FastGFile*) provided by TensorFlow Lite. The app makes inferences on volunteers' smartphones over 7 days to test our model's accuracy. We also collect the status of smartphone usage, i.e., energy consumption and memory usage, which are used to analyze the system overhead.

5.3.1 Performance Analysis. We study the performance gain of *ATPP* from two aspects, i.e., accuracy and latency improvement.

Accuracy. We use the app usage data of volunteers to evaluate the accuracy of *ATPP*, which includes Hirate@5 and MAE. We calculate the accuracy of the *ATPP* every day. For the Hirate@5 metric, we take the average value of all days. For the MAE metric, we calculate not only the average value but also the standard deviation of all days. Figure 16(a) depicts the Hitrate@5 and MAE of all volunteers. It shows that *ATPP* can achieve high performance on average for all volunteers, i.e., 80.2% in Hitrate@5 and 1.48 in MAE. Besides, we can see from the figure that the accuracy of different volunteers fluctuates a little. The reason for the fluctuation might be that the different

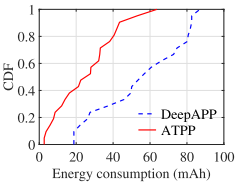


(a) Energy consumption

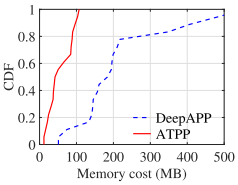


(b) Memory cost

Fig. 17. System overhead of making prediction.



(a) Energy consumption



(b) Memory cost

Fig. 18. System overhead of pre-loading app.

number of apps installed by different users and the frequency of app usage are different, which also proves that *ATPP* is a robust system. Such fluctuating is not caused by the different hardware platforms, because ATPP is a data-driven approach. If the model is trained well, it can be used to make app usage predictions with satisfactory performance regardless of the hardware platform.

Latency improvement. We measure the time reduction in volunteers' smartphones by calculating the average ratio of the saved loading time to the launch time of smartphones without deploying *ATPP*. First, we log the launch time of all apps that are installed on smartphones. We then can obtain the time reduction according to the correctly predicted result of the volunteers. If *ATPP* gives a correct result, then the loading time is zero. Otherwise, it consumes unnecessary energy to load wrong apps into memory and time to release wrong apps from memory and to load correct apps into memory. It ignores the open time of apps if *ATPP* has pre-loaded the apps, for it is neglectable in practice [55]. Figure 16 shows that *ATPP* can reduce the app loading time by 78.1% on average compared with no pre-loading.

5.4 System Overhead

We quantify the overhead produced by *ATPP* from two types: (1) the energy consumption and memory cost of running *ATPP* prediction and (2) the energy consumption and memory cost caused by app pre-loading. To estimate the energy consumption, we first estimate the power consumption rate of each app by a power monitoring application (Accu Battery [44]). It estimates the actual energy consumption based on the information from the battery charge controller by the Android API (BatteryManager Class [4]). It supports obtaining the power consumption using the battery management system in smartphones since Android 5.0. It is widely used in the existing work [2, 27, 43]. Then, we obtain the power consumption of an app based on the app usage time and the power consumption rate of the app.

We compare the overhead of *ATPP* and DeepAPP [43]. To ensure a fair comparison, we let two users use *ATPP* on 2 day and use DeepAPP on another day. Users may perform different app usage activities during 2 days, which means two methods made different numbers of inference. Hence, we use the first 300 pieces of app usage data each day to analyze their performance.

5.4.1 Overhead of Prediction. We measure the overhead on two volunteers who use Samsung Galaxy S9 and Google Pixel 3, respectively. We calculate the average value of these two devices.

Energy consumption. As depicted in Figure 17(a), the extra cost of *ATPP* is about 53.59 mAh on average in a day, which can be almost ignored compared with the total battery capacity of smartphones. DeepAPP consumes about 41.69 mAh on average. Compared with *ATPP*, DeepAPP has less energy consumption. That is because DeepAPP makes prediction inferences on the cloud server, which will save the energy consumption of smartphones, i.e., 12.27 mAh. The energy saved by DeepApp is marginal because DeepAPP needs to communicate with the cloud server in real time. The packet size could be up to 120 bytes.

Memory cost. Figure 17(b) shows the memory cost of *ATPP* and DeepApp. It reveals that the average memory cost of *ATPP* is about 64.15 MB. It means that our system does not consume much extra memory. Memory cost behaves similarly to trends in energy consumption. DeepAPP consumes less memory, i.e., 10.1 MB, because of making inferences on the cloud server. However, current devices, like the Galaxy S9, provide at least 4 GB of memory, which can ignore the slight difference in energy consumption, i.e., 54.05 MB.

5.4.2 Overhead of App Pre-loading. The overhead produced by app pre-loading also mainly contains two aspects, i.e., energy and memory. We also measure the overhead of our system and DeepAPP on volunteers' smartphones.

Energy consumption. Loading apps simultaneously will save more energy than loading apps separately considering that apps may share resources [54]. Therefore, the energy consumption is less than what we measured. Figure 18(a) shows the average energy consumption of *ATPP* and DeepAPP during the experiments. DeepAPP consumes about 87.04 mAh, which accounts for approximately 3% of the energy of the smartphone. However, *ATPP* consumes less than 63.73 mAh of battery energies on average in a day, which is negligible for the total battery energies (e.g., 3,000 mAh). Compared with the DeepAPP, *ATPP* saves 8.9% of energies considering app prediction introduced in Section 5.4.1 and app pre-loading together. The reasons for consuming less energy are as follows.

First, *ATPP* provides higher performance, i.e., 6.0% higher than DeepAPP. Second, *ATPP* pre-loads the app into memory slightly before the user opens it. It minimizes the energy consumption produced by app pre-loading.

Memory cost. We further measure memory usage on smartphones. We monitor the memory usage of participants and get results in Figure 18(b). As shown, our system does not consume much memory on average, i.e., 89.76 MB of total memory. That is because the background scheduler only pre-loads apps when the current time is close to the predicted app open time. Moreover, if the prediction result is wrong, we will immediately free the memory of the app. Note that our system consumes less memory than DeepAPP (i.e., 221.43 MB) because DeepAPP pre-loads all apps that users will be using in the next time slot. If the pre-loaded app is not opened in the current time slot, then it will consume much unnecessary memory, i.e., 131.83 MB. Compared with DeepAPP, *ATPP* can save at least 33.5% of memory in total.

Conclusion. *ATPP* consumes 117.32 mAh (53.59 mAh + 63.73 mAh) of energy and 153.91 MB (64.15 MB + 89.76 MB) of memory. For DeppAPP, it consumes 128.73 mAh (41.69 mAh + 87.04 mAh) of energy and 231.53 (10.1 MB + 221.43 MB) of memory. *ATPP* can reduce energy consumption by 8.9% ((128.73 − 117.32)/128.73) and memory cost by 33.5% ((231.53 − 153.91)/231.53) compared to DeepAPP.

6 RELATED WORK

The latest work related to *ATPP* is DeepAPP [43]. It predicts the apps that a user will open in the next time slot based on deep reinforcement learning [10, 11, 41]. As in [43], the time slot is set to 5 minutes. DeepAPP has to pre-load the next app much earlier before the user opens it, which imposes high memory and energy consumption. Moreover, since DeepAPP models app usage behavior as a one-order Markov Decision Process, it only considers the last app usage event for app prediction, which ignores two key observations made in this article; i.e., next app usage is determined by a number of apps used previously and the last app may not determine next app usage. *ATPP* adopts a totally different approach to predict the next app and its open time accurately. (1) *ATPP* models app usage as an MTPP process that can well capture the temporal dynamics of app usage behavior for app open time prediction. (2) *ATPP* leverages the RNN-based neural network to

accurately learn an MTPP model for each user. (3) *ATPP* integrates the attention mechanism into the RNN-based app prediction framework to consider the influence of sequential apps on current prediction.

AppUsage2Vec [60] adopts an attention mechanism to fuse three types of app usage features, including app usage sequences, user ID, and discrete temporal information. However, both works only consider the prediction of the next app, without the open time of the next app. In addition, although *ATPP* also adopts an attention mechanism, it is totally different from the attention mechanism used in AppUsage2Vec. First, AppUsage2Vec only treats the temporal feature as one factor affecting the next app usage. Its temporal feature specifies the current time in a day and the date in a week. Unlike AppUsage2Vec, *ATPP* captures the temporal dynamics of app usage by the conditional intensity function of the MTPP process. Second, AppUsage2Vec simply leverages the attention mechanism to obtain a weighted summation of all the apps in an app usage sequence. *ATPP* leverages the RNN-based model to learn a set of hidden states that capture the relationship between historical app usage data and the next app usage event.

Besides the above three latest works, conventional methods, like Markov and Bayesian models, have also been widely used for app prediction [6, 12, 26, 28, 38]. Huang et al. [26] incorporate a set of context information, including last used app, time, and location, into a first-order Markov model. Do and Gatica-Perez [12] predict where and which apps a user may use in the next 10 minutes by exploiting the rich contextual information from smartphone sensor data in a Bayesian framework. Chen et al. [6] consider rich context information (e.g., location, time, and app type) and build the user's dynamic profile by graph embedding for personalized app prediction. The model of iConRank [35] first clusters users using a cluster-level Markov model and then calculates a personalized vector from the current context based on the cluster Markov graph. Baezayates et al. [3] combine various explicit features, such as location semantics, and implicit features, including app usage information. These works focus on the prediction of the next app but ignore the prediction of the next app's open time.

Marked temporal point processes. MTPP [1] has been used as a mathematical abstraction to model various phenomena across a wide range of applications, such as human social activities [18], online-user engagement [16], traffic patterns in data center networks [39], event clustering in GitHub [29], localization [49], and packet arrivals in sensor networks [19, 30, 31, 40]. This article extends the application of MTPPs to the app prediction problem with a set of novel techniques. RMTPP [13] uses a neural network to model the intensity function of the subject time-series events. This work extends the application of MTPPs to app prediction by introducing a set of novel techniques. First, *ATPP* integrates a context-aware optimization module into the RNN-based prediction framework to handle drop-in apps and spatial-related app usage patterns. Second, we develop an app representation module to effectively capture the correlation between similar apps.

Cellular data. There are some studies using the cellular network request data [46, 48, 52, 57]. SAMPLES [52] provides a framework to identify the application identity according to the network request by inspecting the HTTP header. Yu et al. [57] present a city-scale analysis of app usage data on smartphones. Tu et al. [46] re-identify a user in the crowd by the apps he or she uses and quantify the uniqueness of app usage. Wang et al. [48] discover users' identities in multiple cyberspaces. However, the above studies do not leverage the app usage data for the app prediction problem.

DNN inference on the edge devices. There are some systems that are focused on the DNN inference on the edge devices [24, 25, 45, 53, 61]. For example, FedDL [45] provides a federated learning system for human activity recognition that can capture the underlying user relationships and apply them to learn personalized models for different users dynamically. DeepCOD [53] has a performance predictor and a runtime partition decision maker to find the optimal partition point

for offloading. These works focus on the inference for images or videos, which needs large-size models, not small-scale models.

7 CONCLUSION

In this work, we develop a novel data-driven MTPP-based app prediction system, named *ATPP*, which can accurately predict both the next app ID and its open time. *ATPP* adopts recurrent neural networks to implement MTPP modeling for app prediction. We incorporate two unique app usage behavior patterns into *ATPP*, i.e., temporal and spatial dependency in app usage. A set of techniques are developed, including an app representation, an app usage event predictor, and a context-aware optimization module. Extensive experiments demonstrate the effectiveness of *ATPP*.

REFERENCES

[1] Odd O. Aalen, Ørnulf Borgan, and Håkon K. Gjessing. 2008. *Survival and Event History Analysis: A Process Point of View*. Springer Science & Business Media.

[2] A. Ahmad, I. Alseadoon, A. Alkhalil, and K. Sultan. 2019. A framework for the evolution of legacy software towards context-aware and portable mobile computing applications. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP'19)*.

[3] Ricardo Baezayates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting the next app that you are going to use. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*.

[4] BatteryManager Class. 2022. TensorFlow Lite. https://developer.android.com/reference/android/os/BatteryManager.

[5] Xiaomeng Chen, Abhilash Jindal, Ning Ding, Yu Charlie Hu, Maruti Gupta, and Rath Vannithamby. 2015. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*.

[6] Xinlei Chen, Yu Wang, Jiayou He, Shijia Pan, Yong Li, and Pei Zhang. 2019. CAP: Context-aware app usage prediction with heterogeneous graph embedding. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 1 (2019), 4.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[8] J. Cleary and I. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications* 32, 4 (1984), 396–402.

[9] Xianzhong Ding and Wan Du. 2022. DRLIC: Deep reinforcement learning for irrigation control. In *Proceedings of the 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'22)*.

[10] Xianzhong Ding, Wan Du, and Alberto Cerpa. 2019. OCTOPUS: Deep reinforcement learning for holistic smart building control. In *Proceedings of the 6th ACM International Conference on Systems for Energy-efficient Buildings, Cities, and Transportation (BuildSys'19)*.

[11] Xianzhong Ding, Wan Du, and Alberto E. Cerpa. 2020. MB2C: Model-based deep reinforcement learning for multi-zone building control. In *Proceedings of the 7th ACM International Conference on Systems for Energy-efficient Buildings, Cities, and Transportation (BuildSys'20)*.

[12] Trinh Minh Tri Do and Daniel Gatica-Perez. 2014. Where and what: Using smartphones to predict next locations and applications in daily life. *Pervasive & Mobile Computing* 12, 10 (2014), 79–91.

[13] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (SIGKDD'16)*.

[14] Shanshan Duan, Weidong Yang, Xuyu Wang, Shiwen Mao, and Yuan Zhang. 2021. Temperature forecasting for stored grain: A deep spatio-temporal attention approach. *IEEE Internet of Things Journal* 8, 23 (2021), 17147–17160.

[15] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[16] Mehrdad Farajtabar, Nan Du, Manuel Gomez Rodriguez, Isabel Valera, Hongyuan Zha, and Le Song. 2014. Shaping social activity by incentivizing users. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS'14)*.

[17] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. Deepmove: Predicting human mobility with attentional recurrent networks. In *Proceedings of the 2018 World WideWeb Conference (WWW'18)*.

[18] Alceu Ferraz Costa, Yuto Yamaguchi, Agma Juci Machado Traina, Caetano Traina Jr., and Christos Faloutsos. 2015. RSC: Mining and modeling temporal activity in social media. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'15)*.

[19] Orestis Georgiou and Usman Raza. 2017. Low power wide area network analysis: Can LoRa scale? *IEEE Wireless Communications Letters* 6, 2 (2017), 162–165.
[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
[21] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).
[22] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y. Chen. 2021. LegoDNN: Block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking (MobiCom'21)*.
[23] Alan G. Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.
[24] Wenchen He, Shaoyong Guo, Song Guo, Xuesong Qiu, and Feng Qi. 2020. Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT. *IEEE Internet of Things Journal* 7, 10 (2020), 9241–9254.
[25] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *2019 IEEE Conference on Computer Communications (INFOCOM'19)*.
[26] Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen. 2012. Predicting mobile application usage using contextual information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (Ubicomp'12)*.
[27] Jude Vivek Joseph, Jeongho Kwak, and George Iosifidis. 2019. Dynamic computation offloading in mobile-edge-cloud computing systems. In *2019 IEEE Wireless Communications and Networking Conference (WCNC'19)*.
[28] Zhung Xun Liao, Shou Chung Li, Wen Chih Peng, Philip S. Yu, and Te Chuan Liu. 2014. On the feature discovery for app usage prediction in smartphones. In *2013 IEEE 13th International Conference on Data Mining (ICDM'14)*.
[29] Shengzhong Liu, Shuochao Yao, Dongxin Liu, Huajie Shao, Yiran Zhao, Xinzhe Fu, and Tarek Abdelzaher. 2019. A latent Hawkes process model for event clustering and temporal dynamics learning with applications in GitHub. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*.
[30] Tao Liu and Alberto E. Cerpa. 2014. Data-driven link quality prediction using link features. *ACM Transactions on Sensor Networks* 10, 2 (2014), 1–35.
[31] Tao Liu and Alberto E. Cerpa. 2014. Temporal adaptive link quality prediction with online learning. *ACM Transactions on Sensor Networks* 10, 3 (2014), 1–41.
[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[33] Mobile app usage report. 2015. App Usage Report. https://www.built.io/assets/blt92f2cb24d08372ae/optimize-mobile-kpis-with-user-centered-metrics-builit.io-whitepaper.pdf.
[34] Mobile deep learning framework. 2021. TensorFlow Lite. https://tensorflow.google.cn/lite/.
[35] Nagarajan Natarajan, Donghyuk Shin, and Inderjit S. Dhillon. 2013. Which app will you use next? Collaborative filtering with interactional context. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys'13)*.
[36] Feiping Nie, Heng Huang, Xiao Cai, and Chris H. Ding. 2010. Efficient and robust feature selection via joint L2, 1-norms minimization. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems (NeurIPS'10)*.
[37] Online map. 2021. AMAP. https://www.amap.com.
[38] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. 2013. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'13)*.
[39] Avirup Saha, Niloy Ganguly, Sandip Chakraborty, and Abir De. 2019. Learning network traffic dynamics using temporal point process. In *2019 IEEE Conference on Computer Communications (INFOCOM'19)*.
[40] Yaman Sangar and Bhuvana Krishnaswamy. 2020. WiChronos: Energy-efficient modulation for long-range, large-scale wireless networks. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom'20)*.
[41] Zhihao Shen, Wan Du, Xi Zhao, and Jianhua Zou. 2020. DMM: Fast map matching for cellular data. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom'20)*.
[42] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, and Jianhua Zou. 2019. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys'19)*.
[43] Zhihao Shen, Kang Yang, Zhao Xi, Jianhua Zou, and Wan Du. 2023. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. *IEEE Transactions on Mobile Computing* 22, 2 (2023), 824–840.
[44] Track your battery's health. 2021. Accu Battery. https://www.accubatteryapp.com/.
[45] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. 2021. FedDL: Federated learning via dynamic layer sharing for human activity recognition. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys'21)*.

[46] Zhen Tu, Runtong Li, Yong Li, Gang Wang, Di Wu, Pan Hui, Li Su, and Jin Depeng. 2018. Your apps give you away: Distinguishing mobile users by their app usage fingerprints. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (2018), 1–23.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS'17)*.

[48] Huandong Wang, Chen Gao, Yong Li, Zhili Zhang, and Depeng Jin. 2017. From fingerprint to footprint: Revealing physical world privacy leakage by cyberspace cookie logs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*.

[49] Xiangyu Wang, Xuyu Wang, Shiwen Mao, Jian Zhang, Senthilkumar C. G. Periaswamy, and Justin Patton. 2020. Indoor radio map construction and localization with deep Gaussian processes. *IEEE Internet of Things Journal* 7, 11 (2020), 11238–11249.

[50] Xu Wang, Zimu Zhou, Fu Xiao, Kai Xing, Zheng Yang, Yunhao Liu, and Chunyi Peng. 2018. Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing* 18, 9 (2018), 2190–2202.

[51] Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. 2021. ATPP: A mobile app prediction system based on deep marked temporal point processes. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS'21)*.

[52] Hongyi Yao, Gyan Ranjan, Alok Tongaonkar, Yong Liao, and Zhuoqing Morley Mao. 2015. SAMPLES: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*.

[53] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys'20)*.

[54] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys'18)*.

[55] Xu Ye, Lin Mu, Lu Hong, Giuseppe Cardone, Nicholas Lane, Zhenyu Chen, Andrew Campbell, and Tanzeem Choudhury. 2013. Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns. In *Proceedings of the 2013 International Symposium on Wearable Computers (ISWC'13)*.

[56] Juheon Yi, Sunghyun Choi, and Youngki Lee. 2020. EagleEye: Wearable camera-based person identification in crowded urban spaces. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom'20)*.

[57] Donghan Yu, Yong Li, Fengli Xu, Pengyu Zhang, and Vassilis Kostakos. 2018. Smartphone app usage prediction using points of interest. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), 1–21.

[58] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2224–2287.

[59] Feixiong Zhang, Chenren Xu, Yanyong Zhang, K. K. Ramakrishnan, Shreyasee Mukherjee, Roy Yates, and Thu Nguyen. 2015. Edgebuffer: Caching and prefetching content at the edge in the mobility first future internet architecture. In *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM'15)*.

[60] Sha Zhao, Zhiling Luo, Ziwen Jiang, Haiyan Wang, Feng Xu, Shijian Li, Jianwei Yin, and Gang Pan. 2019. AppUsage2Vec: Modeling smartphone app usage for prediction. In *2019 IEEE 35th International Conference on Data Engineering (ICDE'19)*.

[61] Yang Kang and Du Wan. 2022. LLDPC: A low-density parity-check coding scheme for LoRa networks. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys'22)*.

Received 5 November 2021; revised 16 July 2022; accepted 15 December 2022