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

Kang Yang[†], Xi Zhao[§], Jianhua Zou[‡], Wan Du[†]
[†] Department of Computer Science and Engineering, University of California, Merced, USA
[§] School of Management, Xi'an Jiaotong University, China
[‡] School of Electronic and Information Engineering, Xi'an Jiaotong University, China
{kyang73, wdu3}@ucmerced.edu, Zhaoxi1@mail.xjtu.edu.cn, jhzou@sei.xjtu.edu.cn

***Abstract*—Predicting the app that a user will open next is essential for improving user experience, *e.g.*, app pre-loading. Unlike previous solutions that only predict next app's ID, this work also predicts the time to open next app. Time prediction is important to avoid loading the next app too early, consuming too much memory and energy on smartphones. To predict next app's ID and open time jointly, we model app usage 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 as *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 events by an attention mechanism, and the location-related app usage behavior by a feature extraction and fusion layer. We conduct extensive experiments on a large-scale anonymized app usage dataset from 443 users over 21 days. The experiment results demonstrate that *ATPP* outperforms the state-of-the-art app prediction method by 6.0% in the prediction accuracy of next app ID and 2.09× reduction in the prediction error of next app open time. A field experiment of 22 users reveals that *ATPP* can reduce the app loading time by 78%.**

***Index Terms*—Mobile Devices, App Usage Prediction, Marked Temporal Point Process, Neural Networks**

I. INTRODUCTION

One effective way to improve mobile user's experience is to minimize app loading time, i.e., pre-loading next app into memory before the user clicks on it [1]–[3]. The key research problem is to predict which app the user will click on next. Many solutions [4]–[9] have been proposed; however, they are mainly focused on predicting the ID of next app, but ignoring the time that the user will open that app. Time prediction is critical for app pre-loading, since it may consume too much energy and memory on smartphones unnecessarily if the predicted apps are loaded too early [10].

This paper proposes to jointly predict the ID of next app and its open time by modeling app usage sequences as a Marked Temporal Point Process (MTPP) [11], whose conditional intensity function can quantify the probability of a new app usage event conditioned by previous events. In our preliminary study, we implement two widely-used MTPP intensity functions, *i.e.*, the Hawkes process [12] and the Homogeneous Poisson process [11]. However, our experiments on an anonymized app usage dataset of 443 users show that such a simple MTPP-based method provides low prediction performance. First, it is hard to find a proper MTPP intensity function to explicitly capture the influence of previous app usage events. Second, app prediction needs to consider some unique features, including the correlated usage behavior of similar apps, the temporal dependency of app usage events, and the location-related app usage behavior.

To address the above limitations, we develop a novel data-driven MTPP-based app prediction system, named as *ATPP* (App Temporal Point Process). It uses Recurrent Neural Network (RNN) [13] to learn the conditional intensity function of MTPP model based on historical app usage data. We implement the RNN model by a set of Gated Recurrent Units (GRU) [14], 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* adopts 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 Hadamard product [15] in the RNN framework for predicting next app ID and its open time. To perform Hadamard product, we adopt a 3-layer fully-connected neural network activated by a $tanh$ function to convert two feature vectors into the same size.

The temporal app usage pattern is that historical app usage events have different influence on the usage of next app. Taking the app 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. To minimize the impact of drop-in apps, we incorporate an attention mechanism [16] 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.

The spatial app usage pattern is that app usage behavior is highly related to spatial context [8]. We develop 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 Interests (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 of POI vectors at different places.

To accelerate the learning process, we develop an efficient app representation module. The input to 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 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 [17], an open source machine learning platform. We train a set of *ATPP* parameters, including the GRU's parameters of the app predictor, and the weight matrix of the app representer and the spatial context feature extractor. We perform end-to-end training of all these parameters by using the Adam algorithm [18], which calculates the gradient of a loss function and updates all learning parameters accordingly.

We conduct both data-driven validation and field experiments. The data-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 minutes 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, *ATPP* can reduce energy consumption by 8.9% and memory cost by 33.5% on smartphones.

In summary, this paper makes the following contributions.

- We model the app prediction problem as a MTPP process for jointly predicting both next app and its open time.
- To transform the above idea into a practical system, we develop *ATPP* with a set of customized techniques, including RNN-based MTPP model learning, app usage event presentation, and temporal and spatial context feature extraction and fusion.
- We conduct both extensive simulations on an app usage dataset and a field experiment. The results demonstrate that *ATPP* outperforms state-of-the-art methods.

II. MTPP-based App Prediction

We model the app usage sequence as a MTPP process [11], which is a random process generated by a sequence of time-series events.

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

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 (the user opens an app) in time window $[t, t+dt)$ given the historical events $\mathcal{H}$.

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

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

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

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 [12] to model app usage behavior, which has been used in many applications to model time sequence data. In Hawkes process, the intensity function $\lambda^*(t)$ is defined as follows.

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

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.

$$\kappa(t) := \alpha\omega \exp(-\omega t) \quad (5)$$

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.

Prediction of next app. We assume app ID usage as an multinomial distribution [11], *i.e.*, the probability that app a_i will be used in next time window $[t, t+dt)$ is determined by its usage frequency in the historical events $\mathcal{H}$.

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

where K is the number of apps installed on the user's smartphone, and f_{a_i} is the probability that app a_i is opened.

Prediction of next app open time. We can use Equation 7 to predict the open time of next app usage [11].

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

where $\lambda^*(t) \exp\left(-\int_{t_i}^{t} \lambda^*(\tau) d\tau\right)$ is the probability density function. It represents the likelihood that an app usage event will occur at the time t given the history.

The above MTPP model needs to learn three parameters, denoted as $\theta = (\mu, \alpha, \omega)$, by app usage data. The learning process is introduced in Appendix A.

Once we have the learned MTPP model, we test its performance by some preliminary experiments on a large dataset of 2,104,369 app usage records from 443 users collected over 21 days by a mobile operator. Smartphones may send requests to a cellular tower when the users click on an app. The cellular tower then sends URLs to the operator server, which parses URLs into corresponding app IDs by url-app encoding tables. We know the user's rough location area based on the location of the associated cell tower.

Our experiment results show that the above MTPP-based solution provides limited performance, *e.g.*, the mean absolute error of time prediction is 4.36 minutes. More detailed experiment results and corresponding analysis can be found in Section IV. The MTPP-based method makes strong assumptions about the generation process of app usage behavior by a specific intensity function. However, the probability of app usage is diverse at different hours, and the app usage patterns of two users are different over the same hour. It is difficult for a specific MTPP process to fit the real app usage behavior for different time and users.

Moreover, the above MTPP-based solution cannot consider temporal or spatial app usage pattern for its predefined simple model. For example, some apps are opened arbitrarily when users are using their phone. So historical app usage events may have different influences on the current prediction due to temporal app usage behavior. Or environment context (*e.g.*, spatial context) has a significant influence on the apps usage behavior [8].

III. Design of *ATPP*

In this section, we introduce the design of *ATPP* and its key components to handle the above challenges.

A. Architecture of ATPP

Figure 1 depicts the architecture of *ATPP*, consisting of three key modules, *i.e.*, an app representation, 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 to convert it into a sequence of

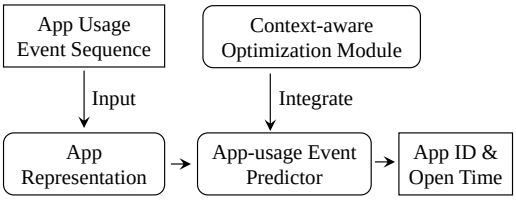


Fig. 1. The architecture of *ATPP*.

high-quality representations for each app usage event (Section III-B). Then, an RNN-based app-usage event predictor is applied to predict the app ID that user most-likely use and its open time (Section III-C). 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 context feature extractor (Section III-D).

The above prediction (inference) process includes three neural networks, including an app representation (a two-layer neural network), an RNN-based app predictor and a context-aware optimization module. We perform end-to-end training for all these neural networks at the same time. We design a loss function (Appendix C) 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.

B. 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 2, 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 "0", especially when a user installs a large number of apps on 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 to automatically learn a low-dimensional representation for each app usage event.

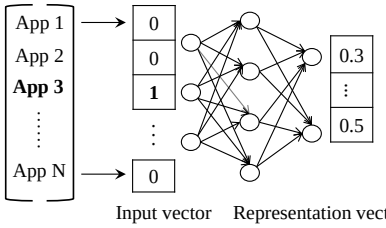


Fig. 2. The architecture of app representation.

Figure 2 depicts the architecture of the app representation 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.

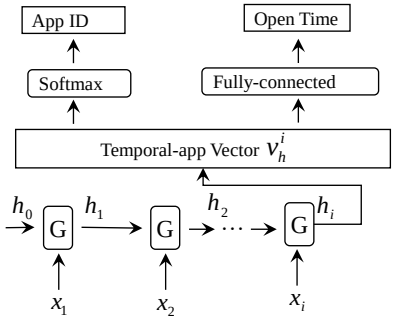


Fig. 3. The architecture of the app-usage event predictor.

Specifically, the inter-event duration between the i-th 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 current 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 predictor.

C. App-usage event predictor

We design a recurrent MTPP-based app prediction system, which leverages an RNN model to learn the intensity function of MTPP process in a data-driven manner. Figure 3 shows the architecture of our app-usage event predictor. A recurrent neural network composed of Gated Recurrent Units (GRU) is 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 next app ID and to a fully-connected layer for predicting open time. N is the length of input app sequence. In our current implementation, N is set to 5. The input of 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 [14]. Although Long Short-Term Memory (LSTM) [19] is also widely used, GRU achieves similar performance in many tasks with less computation [14]. The computation of GRU can be expressed as follows.

$$\mathbf{h}_i = \mathrm{GRU}\left(x_i, \mathbf{h}_{i-1}\right) \tag{8}$$

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

At the beginning of the training process, $\mathbf{h_0}$ is uniformly initialized to random values $[-0.1, 0.1]$ for the first app sequence. To be consistent, in the rest of paper, the current hidden state $\mathbf{h_i}$, is also referred as the temporal app vector $\mathbf{v}_h^i$, *i.e.*, $\mathbf{v}_h^i = \mathbf{h_i}$. Based on the temporal app vector, we predict 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 as follows.

$$\hat{a}_{i+1} = \mathrm{argmax}\left\{ \mathit{softmax}\left(\mathbf{V}_s \cdot \mathbf{v}_h^i\right)\right\} \tag{9}$$

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 a MTPP conditional intensity function via the RNN output $\mathbf{v}_h^i$. Inspired by RMTPP [20], we use the learned hidden states to calculate a general representation of intensity function. Compared to simple MTPP-based app prediction method that using a predefined intensity function

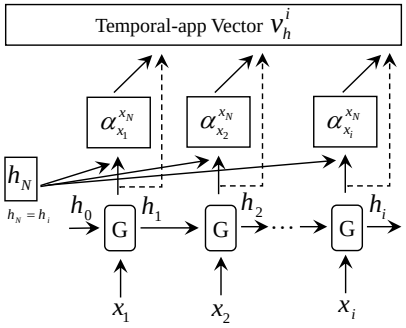


Fig. 4. The architecture of attention-based feature extractor.

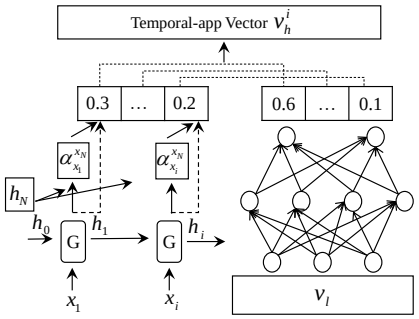


Fig. 5. Combination of the results from two feature extractors.

(*e.g.*, Equation 4), Equation 10 provides a general representation of intensity function that can be learned from historical data.

$$\lambda^*(t) = \exp\left(\mathbf{v}_t^\top \cdot \mathbf{v}_h^i + w_t \cdot (t - t_i) + b_t\right) \tag{10}$$

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 10, 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 next app. The second part, $w_t \cdot (t - t_i)$, emphasizes on the influence of the latest app usage event (t_i, a_i). The last term b_t represents a base intensity level. The exponential function provides a non-linear transformation that makes the intensity function positive.

Based on the conditional intensity function obtained from Equation 10, we can predict next app open time $\hat{t_{i+1}}$ through Equation 7.

D. 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 temporal feature extractor and a spatial context feature extractor. We update the temporal-app vector $\mathbf{v}_h^i$ by fusing the above two features.

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 4 depicts the architecture of attention-based temporal feature extractor, which integrates soft attention mechanism [16] into *ATPP* to handle drop-in app usage, which calculates a vector as 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 generated a fused vector in this section. we can obtain a new temporal-app vector $\mathbf{v}_h^i$ as follows.

$$\mathbf{v}_h^i = \sum_{i=1}^{N} \alpha_{x_i} \mathbf{h}_i \tag{11}$$

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

$$\alpha_{x_i} = |\tanh(\mathbf{h}_i * \mathbf{h}_N)| \tag{12}$$

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. One app usage event with a higher weight should be paid to more attention. We normalize all the weights to make sure that they are summed to 1.

2) Spatial context feature extractor: A user may have diverse app usage behavior in different spatial contexts [8]. So we develop a spatial context feature extractor to incorporate the spatial context into our app predictor. Figure 5 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 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 the spatial information for a user. To this end, we leverage the point of interest (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, supermarkets, and so on.

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 this 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 [21], which provides application programming interface (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 POI. The dimension corresponds to the number of categories of POI, where l^j is the POI number of type j within the radius of 500 meters.

3) Feature fusion: We apply two methods to combine the spatial context feature vector $\mathbf{v}_l$ and the temporal-app vector $\mathbf{v}_h^i$ obtained from the attention-based temporal 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 [15] to perform element-wise multiplication of these two feature vectors.

The Hadamard product requires two feature vectors 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 temporal feature extractor.

Based on our experiments (see Figure 8 in Section IV-B2), we find Hadamard product provides better performance in our app prediction. Using the Hadamard product, we obtain a new temporal-app vector $\mathbf{v}_h^i$. We can use the new vector to predict next app and its open time, as introduced in Section III-C.

IV. Evaluation

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

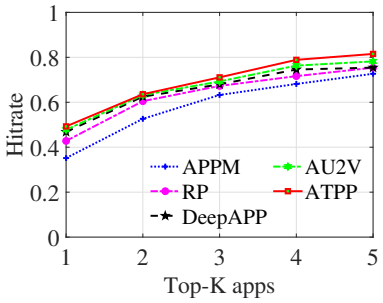


Fig. 6. Performance criteria : Hitrate.

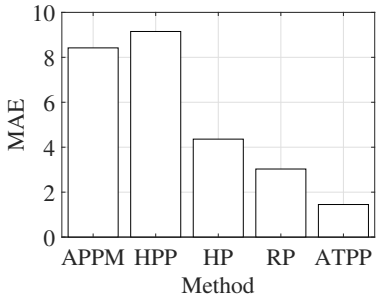


Fig. 7. Performance criteria : MAE.

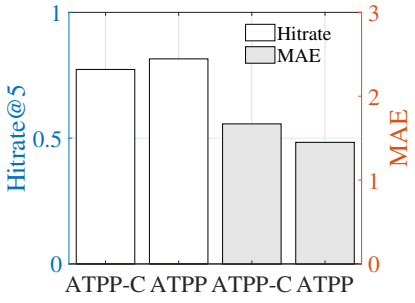


Fig. 8. The gain of Hadamard product.

A. Experiment Settings

Performance criteria. To evaluate the performance of *ATPP*, we use two metrics: Hitrate@K and Mean Absolute Error (MAE).

Hitrate@K calculates the hit ratio of the top-K predicted apps, which is used to evaluate the accuracy of next app prediction. Users can only open one app each time. When a user clicks anyone of the top-K predicted apps, we consider it as one hit.

MAE is used to evaluate the accuracy of app open time prediction. It measures the absolute difference between the predicted timestamp and the ground truth. Its unit is minutes. When we calculate MAE, we ignore whether the predicted next apps are correct.

Benchmarks. We compare the performance of *ATPP* with two types of existing solutions. First, DeepAPP [8] and AU2V [9] are based on deep learning model [22] that predicts next app. Second, APPM [5] 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 next app and open time, including Homogeneous Poisson Process-based method (HPP), Hawkes Process-based method (HP), app-usage event predictor (RP).

- *DeepAPP [8].* It predicts the apps that a user will open in the next time slot based on the reinforcement learning, which considers two context information, including the last used app and spatial feature. We set the time slot as one minute in our implementation.
- *AU2V [9].* It incorporates the app sequence, user personalized characteristics, and discrete temporal context to predict next app that she most likely opens.
- *APPM [5].* It uses the app sequence to compute the probability of the following app based on Prediction by Partial Match Model (PPM) [5]. Moreover, it predicts app open time through a Time Till Usage (TTU) model.
- *HPP [11].* We model app usage behavior as Homogeneous Poisson process (HPP). Its intensity function is constant. It is the simplest MTPP-based method to model the app usage behavior.
- *HP [12].* It assumes that app usage behavior as the Hawkes process which is introduced in Section II.
- *RP.* Similar to RMTPP [20], RP simply uses an RNN to learn a MTPP conditional intensity function for app predicition.

Evaluation setup. We compare *ATPP* with above baselines in Section IV-B1, then evaluate the effectiveness of Hadamard product in Section IV-B2. We also verify the effectiveness of the context-aware optimization module in Section IV-B3. The default value of parameters is further discussed in Section IV-B4. Next, we do field experiments with 22 users in Section IV-C, including model's accuracy and latency improvement of each user. Finally, we compare the overhead produced by *ATPP* and DeepAPP in Section IV-D.

B. Data-driven Evaluation

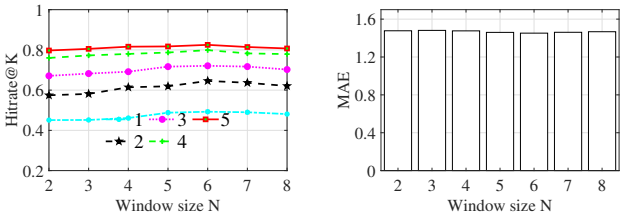
We conduct data-driven evaluations on a large-scale anonymized app usage dataset introduced in Section II. We divide the dataset into two parts, *i.e.*, 14-day data for training and 7-day data for testing. We set the default value of the size of the time window N to 5 and the dimension of the temporal-app vector D to 64. We use these settings by default to conduct following experiments. In Section IV-B4, we will explain how we set these parameters to the optimal values. And the implementation details of *ATPP* model is introduced in Appendix B.

1) Overall performance comparison: As shown in Figure 6, *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, 3.3% higher than AU2V. The reason for such improvement is that the attention-based temporal feature extractor captures the weithted influence of historical app usage events sequences for prediction task. And the spatial context feature extractor denotes personalized app usage patterns on some specific locations.

APPM gives the worst performances, for it just utilizes the app sequences information through the PPM [5] 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 longer app usage sequence through a recurrent neural network. DeepAPP gives a little worse 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 that users opened 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. But, it can not be used efficiently in practice for it does not support predicting open time.

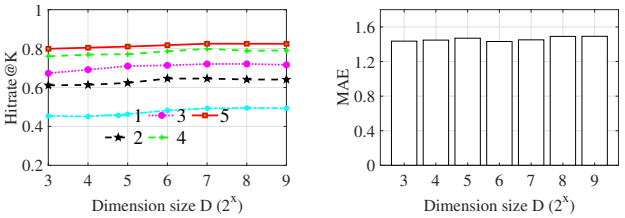
Although *ATPP* may provide a marginal improvement on the prediction of next app, *ATPP* can accurately predict app open time. As shown in Figure 7, *ATPP* performs best in MAE. *ATPP* achieves a significant improvement in MAE, making it around 6.31× reduction than HPP, 3.01× reduction than HP, 5.80× reduction than APPM, and 2.09× reduction than RP. The reason for making such an improvement is that simple MTPP-based models make strong assumptions about app usage behaviors. Although RP leverages a recurrent neural network to model intensity function, it doesn't consider drop-in app usage and spatial-related app usage bahavior.

2) Performance gain of Hadamard product: There are two methods to combine the attention-based temporal feature extractor and the spatial context feature extractor, *i.e.*, concatenation or Hadamard product [15]. Figure 8 presents the performance of the two methods.



(a) Hitrate@K (b) Mean absolute error (MAE)

Fig. 9. Effect of the window size N.

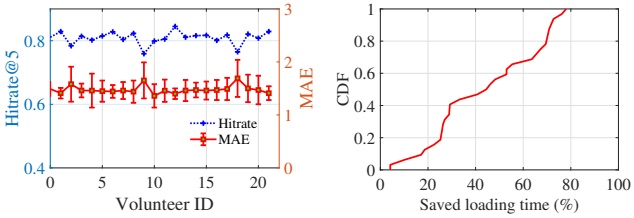


(a) Hitrate@K (b) Mean absolute error (MAE)

Fig. 10. Effect of the dimension of vector D.

TABLE I
EFFECTIVENESS OF PROPOSED MODULES.

Model	Hitrate@5	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×



(a) Accuracy (b) Latency improvement

Fig. 11. Accuracy & latency improvement analysis.

The 'ATPP-C' denotes the operation of concatenation. The Hadamard product gives better performance than concatenation. Hence, we choose Hadamard product to integrate two feature extractors.

3) Effectiveness of two modules in context-aware optimization: We investigate the improvement on the performance of the two feature extractors from 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 I. 'RP+AF' indicates app-usage event predictor with the module of *AF*. 'RP+SF' means predictor with the module of *SF*, which directly integrates last hidden state with spatial vector through the Hadamard product. 'HGain' is the gain of Hitrate@5, 'MGain' is the gain of MAE.

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

Attention-based temporal feature extractor. This component improves the prediction performance around 4.8% higher than the baseline in Hitrate@5, and 1.79× reduction in MAE. The module incorporates drop-in app usage into system, which gives the significant performance improvement in predicting next app usage. It also verifies that drop-in app usage has a great influence on app prediction.

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. It validates the importance of spatial context in the app prediction task.

4) Parameters settings: We further test the choice of two parameters in *ATPP*, *i.e.*, the length of app usage event sequences N, 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 9, 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 probably because inputting such a long app usage event sequence increases the difficulty in training our system. Moreover, if we input longer sequences to predict next app, there are relatively few training data.

This trend of change in MAE is similar to the Hitrate@K. It can be seen 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 also test the performance of *ATPP* with varying temporal-app vectors' dimension D when the window size is 5, shown in Figure 10. Hitrate@1 increases relatively quickly when the dimension 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 dimension size varies from 2^6 to 2^9, and the MAE is best when the dimension size is 2^6.

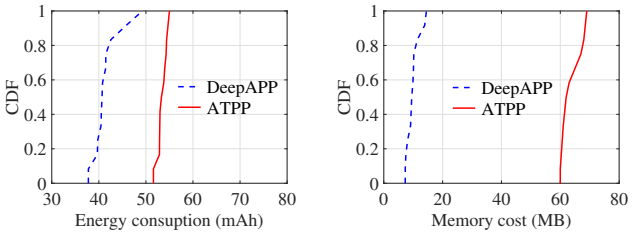
C. Field study

We do field experiments to test *ATPP* from 31 May to 20 June 2020. We deploy the system as the architecture in Figure 1. We recruit 22 volunteers. They include 7 females and 15 males, aged from 17 to 48, which have different occupations, for example, company employees, college teachers, students, and so on. After volunteers agree to experiment, we install the Android app on their smartphones. First, we collect 14 days of data that are used to train the model of volunteers, then load the well-trained model into the app. The app makes inferences on the smartphone of volunteers during the last seven days. We also collect the status of smartphone usage, such as energy consumption and memory usage, which are used to analyze the system overhead.

1) Performance analysis: We explore the performance of *ATPP* from two aspects, *i.e.*, accuracy and latency improvement.

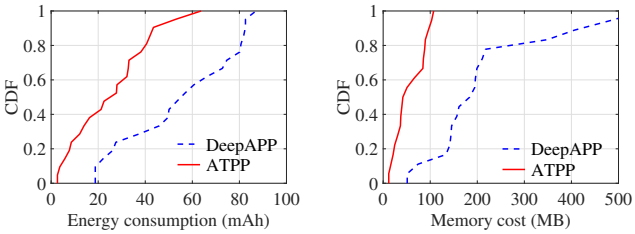
Accuracy. We leverage the app usage data of volunteers to evaluate the accuracy of *ATPP*, which includes Hirate@5 and MAE. We calculate the accuracy of the *ATPP* every day. For Hirate@5 metric, we take the average value of all days. For MAE metric, we not only calculate the average value but also the standard deviation of all days. Figure 11(a) depicts the Hitrate@5 and MAE of all users who participated in this field experiment. 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 is a little fluctuating. The reason for such fluctuation may be that the number of apps installed by different users, and the frequency of app usage are different, which proves that *ATPP* is a robust system.

Latency improvement. We measure the time reduction on volunteers' smartphones by calculating the average ratio of the saved



(a) Energy consumption (b) Memory cost

Fig. 12. System overhead of making prediction.



(a) Energy consumption (b) Memory cost

Fig. 13. System overhead of pre-loading app.

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 [23]. Figure 11(b) shows that *ATPP* can reduce the app loading time by 78.1% on average compared with no pre-loading.

DeepAPP takes the survey to collect the feedback on the usability of the app prediction system from volunteers. It shows that 87.51% volunteers are satisfied with the app prediction system. Compared with DeepAPP, *ATPP* not only provides better performance on next app prediction but also predicts open time. So *ATPP* has a higher probability to provide users with better satisfaction of usability.

D. System Overhead

We quantify the overhead produced by our system from two aspects, *i.e.*, 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 [24]). It estimates the actual energy consumption based on the information from the battery charge controller. Then, we calculate 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 [8]. To ensure a fair comparison, we let two volunteers use *ATPP* on one day and use DeepAPP on another day. Volunteers may perform different app usage activities during two days, which means two methods made different numbers of inference. Therefore, we just use the first 300 pieces of app usage data each day to analyze their performance.

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 12(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 inference 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 up to 120 bytes, which consume energy.

Memory cost. Figure 12(b) shows the memory cost of *ATPP*. 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 similar trends in energy consumption. DeepAPP consumes less memory, *i.e.*, 10.1 MB, because of making inference on the cloud server. However, current devices, like Samsung Galaxy S9, provides at least 4GB memory, which can ignore the slight difference in energy consumption, *i.e.*, 54.05MB.

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. As we known, apps may share resources, so loading apps simultaneously will save energy than loading apps separately [25]. Thus the energy consumption is less than what we measure. Figure 13(a) shows the average energy consumption during the experiment. *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.*, 3000 mAh). However, DeepAPP consumes about 87.04 mAh, which consumes approximately 3% of the energy of the smartphone. Compared with the DeepAPP, *ATPP* can save 8.9% of energies considering inference and app pre-loading together.

The reasons for consuming less energy in our system are as follows. First, ATPP pre-loads the app into memory slightly before the user opens it. It minimizes the energy consumption produced by app pre-loading. Second, *ATPP* provides higher performance, *i.e.*, 6.0% higher than DeepAPP. Compared with DeepAPP, *ATPP* introduces a few additional energy consumption, *i.e.*, 23.31 mAh.

Memory cost. We further measure the memory usage on smartphones. We monitor the memory usage of participants and get results in Figure 13(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 used 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.

V. Related Work

The latest work related to *ATPP* is DeepAPP [8]. It predicts the apps that a user will open in the next time slot (5 minutes in [8]) based on deep reinforcement learning [26]–[28]. DeepAPP has to pre-load 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 paper, *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 next app and its open time accurately. 1) *ATPP* models app usage as a 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 a 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.

Chen *et al.* [7] 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. AppUsage2Vec [9] 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 next app, without the open time of 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 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* leverage 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 [4], [5]. Huang *et al.* [4] incorporate a set of context information, including last used app, time, and location, into a first-order Markov model. These works are mainly focused on the prediction of next app, but ignore the prediction of open time.

Marked temporal point processes. MTPP [11] has been used as a mathematical abstraction to model various phenomena across a wide range of applications, such as human social activities [29]. RMTPP [20] 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 correlation of similar apps.

VI. Conclusion

In this work, we develop a novel data-driven MTPP-based app prediction system, named as *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] "App Usage Report," https://www.built.io/assets/blt92f2cb24d08372ae/optimize-mobile-kpis-with-user-centered-metrics-builit.io-whitepaper.pdf, 2015.

[2] C. Xu, V. Srinivasan, J. Yang, Y. Hirase, E. M. Tapia, and Y. Zhang, "Boe: Context-aware global power management for mobile devices balancing battery outage and user experience," in *IEEE MASS*, 2014.

[3] F. Zhang, C. Xu, Y. Zhang, K. Ramakrishnan, S. Mukherjee, R. Yates, and T. Nguyen, "Edgebuffer: Caching and prefetching content at the edge in the mobilityfirst future internet architecture," in *IEEE WoWMoM*, 2015.

[4] K. Huang, C. Zhang, X. Ma, and G. Chen, "Predicting mobile application usage using contextual information," in *ACM Ubicomp*, 2012.

[5] 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 *ACM Ubicomp*, 2013.

[6] T. M. T. Do and D. Gatica-Perez, "Where and what: Using smartphones to predict next locations and applications in daily life," *Pervasive & Mobile Computing*, vol. 12, no. 10, pp. 79–91, 2014.

[7] X. Chen, Y. Wang, J. He, S. Pan, Y. Li, and P. Zhang, "Cap: Context-aware app usage prediction with heterogeneous graph embedding," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, p. 4, 2019.

[8] Z. Shen, K. Yang, W. Du, X. Zhao, and J. Zou, "Deepapp: a deep reinforcement learning framework for mobile application usage prediction," in *ACM SenSys*, 2019.

[9] S. Zhao, Z. Luo, Z. Jiang, H. Wang, F. Xu, S. Li, J. Yin, and G. Pan, "Appusage2vec: Modeling smartphone app usage for prediction," in *IEEE ICDE*, 2019.

[10] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone background activities in the wild: Origin, energy drain, and optimization," in *ACM MOBICOM*, 2015.

[11] O. O. Aalen, Ørnulf Borgan, and H. K. Gjessing, *Survival and event history analysis : a process point of view*. Springer Science & Business Media, 2008.

[12] A. G. Hawkes, "Spectra of some self-exciting and mutually exciting point processes," *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[15] J.-H. Kim, K.-W. On, W. Lim, J. Kim, J.-W. Ha, and B.-T. Zhang, "Hadamard product for low-rank bilinear pooling," *arXiv preprint arXiv:1610.04325*, 2016.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[17] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

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

[19] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[20] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *ACM SIGKDD*, 2016.

[21] "AMAP," https://www.amap.com, 2021.

[22] M. Liu, X. Ding, and W. Du, "Continuous, real-time object detection on mobiledevices without offloading," in *IEEE ICDCS*, 2020.

[23] X. Ye, L. Mu, L. Hong, 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 *IEEE ISWC*, 2013.

[24] "Accu Battery," https://www.accubatteryapp.com/, 2021.

[25] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *ACM SenSys*, 2018.

[26] Z. Shen, W. Du, X. Zhao, and J. Zou, "Dmm: fast map matching for cellular data," in *ACM MobiCom*, 2020.

[27] X. Ding, W. Du, and A. E. Cerpa, "Mb2c: Model-based deep reinforcement learning for multi-zone building control," in *ACM BuildSys*, 2020.

[28] X. Ding, W. Du, and A. Cerpa, "Octopus: Deep reinforcement learning for holistic smart building control," in *ACM BuildSys*, 2019.

[29] A. Ferraz Costa, Y. Yamaguchi, A. Juci Machado Traina, C. Traina Jr, and C. Faloutsos, "Rsc: Mining and modeling temporal activity in social media," in *ACM SIGKDD*, 2015.

[30] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint l2, 1-norms minimization," in *NeurIPS*, 2010.

APPENDIX A
PARAMETER LEARNING OF THE MTPP MODEL

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

$$L = \left(\prod_{i=1}^{N} \lambda^* (t_i, a_i)\right) \exp\left(-\int_0^T \lambda^*(s)\mathrm{d}s\right) \tag{13}$$

where $\lambda^*(t, a_i)$ is the conditional intensity function for the usage event of app a_i in next time window $[t, t+dt)$. In our implementation, T is one hour. We can calculate $\lambda^*(t, a_i)$ based on Equation 4, 5, 6. The likelihood function is the joint density function of all the app usge events in the observed data $\mathcal{H}$. The last term in Equation 13 represents the probability of no app used at $t \in [0, T]$ $except$ $\{t_i\}$. With the likelihood in Equation 13, we can use our data to find the best setting of the three parameters $\theta = (\mu, \alpha, \omega)$, through MLE.

APPENDIX B
MODEL IMPLEMENTATION

We implement *ATPP* on TensorFlow [17] platform. Three models in *ATPP* are implemented, *i.e.*, an app representer (Section III-B), an app-usage event predictor (Section III-C), and a context-aware optimization module (Section III-D).

First, we implement the app representer as a two-layer neural network, which has K and D neurons, respectively. K is the dimension of 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 IV-B4.

Second, for app-usage event predictor, the length of app usage sequence that used to train model is 5. we also use a two-layer fully-connected feedforward neural networks as the fully-connected layer that used to feed to softmax layer. Specifically, the layer has D and K neurons in the two layers, respectively. We adopt $relu$ activation function for the first layer and $tanh$ activation function for the second layer. Moreover, to alleviate the over-fitting problem when training our system, we introduce an L_2 regularization term [30] in the loss function introduced in Appendix C.

Third, for context-aware optimization, we use a three-layer fully-connected feedforward neural network has 23, $D/2$, and D neurons in the three layers, respectively. Their activation function are $relu$ function, where 23 is the number of categories of POI that provided by AMap [21]. Besides, we conduct grid search strategy to get the optimal settings of hyper-parameters, as shown in Table II.

TABLE II
HYPER-PARAMETER SETTING.

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

APPENDIX C
MODEL TRAINING

In *ATPP*, there are a set of neural network weight parameters to learn, including the weight matrix $\mathbf{W}_{KD}$ of the app representer, the GRU's parameters of the app predictor, and the weight matrix $\mathbf{W}_{EL}$ of the spatial context feature extractor. We maximize the log-likelihood ($\mathcal{LL}$) [20] of observing app usage event sequences $\mathcal{H} = \{t_i, a_i, l_i\}_{i=1}^N$ to train these parameters.

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

where $\mathbf{u}_{a_i} = \textit{softmax}\left(\mathbf{V}_s \mathbf{v}_h^i\right)$ is the probability that all apps are used. The first term represents the probability of 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 Adam algorithm [18], which can be obtained from Equation 14. In this way, we can train *ATPP* model in an end-to-end manner. We first use all users' 14-day data to train a general model and then reuse the 14-day data of each user to further train her own RNN model. Finally, we use the following 7-day data for model validation and evaluation.

Discussion on the dataset. The dataset used in the data-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 during field experiments. It can log all apps' activities installed on the smartphone. Moreover, *ATPP* gives comparable performance in the field study, compared with data-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, 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. In addition, we do not collect precise location of the user for our system only needs the POI distribution information around cell towers.