DeepAPP: A Deep Reinforcement Learning Framework for Mobile Application Usage Prediction

Zhihao Shen, Kang Yang, Xi Zhao, *Senior Member, IEEE*, Jianhua Zou, *Member, IEEE*, and Wan Du, *Member, IEEE*

Abstract—This paper aims to predict a set of apps a user will open on her mobile device in the next time slot. Such an information is essential for many smartphone operations, e.g., app pre-loading and content pre-caching, to improve user experience. However, it is hard to build an explicit model that accurately captures the complex environment context and predicts a set of apps at one time. This paper presents a deep reinforcement learning framework, named as DeepAPP, which learns a model-free predictive neural network from historical app usage data. Meanwhile, an online updating strategy is designed to adapt the predictive network to the time-varying app usage behavior. To transform DeepAPP into a practical deep reinforcement learning system, several challenges are addressed by developing a context representation method for complex contextual environment, a general agent for overcoming data sparsity and a lightweight personalized agent for minimizing the prediction time. Extensive experiments on a large-scale anonymized app usage dataset reveal that DeepAPP provides high accuracy (precision 70.6 percent and recall of 62.4 percent) and reduces the prediction time of the state-of-the-art by 6.58×. A field experiment of 29 participants demonstrates DeepAPP can effectively reduce launch time of apps.

Index Terms—Mobile devices, app usage prediction, deep reinforcement learning, neural networks

1 Introduction

Predicting the next applications (apps) that a mobile user may use in the next time slot can provide many benefits on smartphones, such as app pre-loading [1], [2], [3], content pre-caching [4], [5], [6] and resource scheduling [7]. For instance, by knowing the apps a user may open in next 5 minutes, we can pre-load the apps in memory slightly in advance and improve user experience with minimized launch time. However, most app prediction works have been proposed to predict the next app. They do not consider when to open the apps, and hence cannot be directly used for time-sensitive app prediction systems.

Most existing app prediction works [1], [8], [9], [10], [11], [12], [13], [14], [15], [16] can only provide limited prediction accuracy due to two reasons. First, most conventional model-based methods assume app usages can be well modeled by Markov chain [4], [9], [13], [17] or Bayesian framework [11]. However, app usages are determined by a variety of factors in the complex contextual environment. It is hard to explicitly capture the impact of all potential factors by a statistical model. As a consequence, most existing works [10], [11] represent the context by a limited number of semantic labels (i.e., "Home", "Work place" and "On the way"). Second, people may use a set of apps in a time slot. Different combinations of apps lead to a large number of prediction results. Existing approaches [4], [16] can only calculate probabilities of the apps that are most likely to use separately, and predict a set of apps with highest probabilities. However, these approaches ignore relationship among the predicted apps. In practice, a bundle of apps may receive higher probabilities than predicting the apps separately. For instance, a user may use a shopping app and a payment app in a combination.

To address the above limitations, we develop a Deep Reinforcement Learning (DRL) based framework, named as *DeepAPP*, to learn a data-driven model-free neural network (also known as an agent in DRL), which takes the environment context as input and predicts the apps that will be opened in the next time slot. We first train a deep neural network (DNN) agent using historical app usage data on a server and then run the trained DNN agent on either the server or user's smartphone. The DNN agent of DeepAPP makes prediction based on a neural network rather than an explicit model; therefore, it can take the complex environment context as input. Additionally, with reinforcement learning, DeepAPP learns to explore the large prediction space automatically and effectively select the best list of apps for the prediction result. To incorporate DRL into DeepAPP, we tackle a set of challenges and develop three novel techniques for app prediction,

- *Zhihao Shen and Jianhua Zou are with the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China. E-mail: szh1095738849@stu.xjtu.edu.cn, jhzou@sei.xjtu.edu.cn.*
- *Kang Yang and Wan Du are with the Department of Computer Science and Engineering, University of California Merced, Merced, CA 95340 USA. E-mail: {kyang73, wdu3}@ucmerced.edu.*
- *Xi Zhao is with the School of Management, and the Key Lab of the Ministry of Education for process control & Efficiency Egineering, Xi'an Jiaotong University, Xi'an 710049, China. E-mail: Zhaoxi1@mail.xjtu.edu.cn.*

Manuscript received 12 June 2020; revised 18 May 2021; accepted 25 June 2021. Date of publication 30 June 2021; date of current version 6 January 2023. (Corresponding authors: Xi Zhao and Jianhua Zou.) Digital Object Identifier no. 10.1109/TMC.2021.3093619

including a context-aware DRL input representation method, a lightweight agent and an agent enhancement scheme.

To enable more accurate app prediction, DeepAPP leverages more fine-grained representation of the environment context. Besides the time and currently-opened app [1], [13], DeepAPP leverages the distribution of surrounding Point of Interests (POIs) to capture the location features of the user. In addition, based on such a representation, the DNN-based agent can generalize the past experience to new locations. When a user goes to a new place, the DNN model can still make prediction according to similar known places.

In order to provide real-time inference, one essential requirement of app prediction is short inference latency. One successful implementation of DRL is Deep Q-Network (DQN) [18], [19], which has been applied in many applications, like Atari games and mobile Convolutional Neural Network (CNN) model selection [20]. For one inference, DQN searches for the best action from all possible actions. It is efficient for small action spaces (e.g., 2 actions for Breakout in Atari game), but cannot be used for our app prediction system due to the large action space. For example, if a user has installed 20 apps, the action space will be enormous ($C_{20}^{0} + C_{20}^{1} + \ldots + C_{20}^{20} = 2^{20} = 1,048,576$). DQN-based model will take 2.04 seconds to perform one prediction in our implementation on a 2-core CPU, which is unacceptable in a real-time prediction environment. To handle this problem, we adopt a lightweight actor-critic based architecture [21] to avoid the heavy cost of evaluating all possible actions for one inference.

Ideally, we can train a specific DRL model for each individual user based on her own app usage data. However, it is difficult to obtain sufficient training data from each user. Additionally, users may install new apps [22]. It is hard for a trained agent to cover these new apps during online inference. To solve the data sparsity problem, DeepAPP first trains a general agent with the data of all available users (e.g., 443 users in our dataset). We then use the trained agent for app prediction of every user. As personal app usage data are collected in a sequential manner, we also propose an online learning strategy [23] to keep updating the agent to a personalized agent for each user based on her new app usage data. At the same time, we also update the general agent periodically (e.g., one day in our implementation) using the app usage data from all users. Once the general agent is updated, we also use it to further update each personalized agent by combining their DNN parameters. As each user has increasingly collected her own data to update her personalized agent, an adaptive coefficient is defined to gradually reduce the weight of the general agent in the update of each personalized agent.

We implement DeepAPP on TensorFlow [24]. We run the personalized agents of all users independently on a server that contains 2 CPUs. Experiment results demonstrate that two CPU cores are enough to make an inference within 0.31 second. Although such light cost of inference and agent update can be totally supported by current smartphones, we cannot run the TensorFlow version of DeepAPP on smartphones, since TensorFlow currently does not support the training of the DNN model on mobile operating systems. We further implement DeepAPP on TensorFlow Lite [25] to perform inference directly on smartphones.

We first conduct trace-driven validations. Our dataset contains the app usage records of 21 days from 443 users in a big city. We use the 14-day data for training and the rest 7-day data for validation. Cross-validation tests are conducted. We train the general agent by the training data of all users, and update the personalized agent incrementally for each user using her testing data. The experiment results demonstrate that DeepAPP provides precision and recall of 70.6 and 62.4 percent respectively, corresponding to a performance gain of 8.62 and 15.56 percent over the state-of-the-art solution [16]. DeepAPP also provides a 6.58× inference time reduction compared with the DQN-based model.

We also recruit 29 volunteers and conduct field experiments over 55 days by a customized app. The experiment results reveal that DeepAPP provides precision and recall of 73.2 and 54.1 percent respectively in app prediction. With app pre-loading, DeepAPP can reduce the app loading time by 68.14 percent on average. More than 85 percent of the participants are satisfied with our app prediction system.

In summary, this paper makes following contributions.

- To the best of our knowledge, we are the first to leverage deep reinforcement learning in app prediction.
- We customize our DRL-based framework by considering unique challenges in our app prediction system , including a context representation method, a lightweight personalized agent and an agent enhancement technique by the data of all available users.
- We conduct extensive evaluations based on a large-scale app usage dataset and field experiments.

2 Motivation

In this section, we first investigate the necessity for app prediction through questionnaires. We then introduce the data used in this work for app prediction system. Finally, we briefly introduce the key concepts of deep reinforcement learning.

2.1 Need for App Prediction

We designed and released a questionnaire on a widely used online questionnaire survey platform, called WJX [26]. Questions are mainly about the necessity and urgency of an app prediction system. After 32-day collection, 238 enrolled participants returned their feedback We filtered out invalid feedbacks and eventually we got 206 questionnaires. The participants include 65 females and 141 males, aged from 13 to 65. They have various occupations, such as company employees, civil servants, medical staff, college teachers, students, etc. The survey results indicate an urgent request for accurate app prediction. We have the following detailed analysis of our collected feedback.

- 76.63 percent of them thought it takes a long time from clicking on an application icon to start using the application;
- 90.77 percent of them are willing to use a software that can reduce waiting time of application loading.

2.2 App Usage Dataset

We use an anonymized app usage dataset collected by a mobile carrier of a big city in China. When users request network services from mobile apps, the requests were passively recorded by the cellular infrastructure. The request captures the anonymized identification (ID) of each mobile device, start and end timestamps of the data connection, the HTTP request or response URL (Uniform Resource Locator) and the cell tower from which the request was made. In order to make the data available for our work, we preprocess the data with three steps. First, we infer the apps that are most likely to generate the URLs according to the mapping table of the app and URL provided by the carrier. Second, we obtain the app usage duration by merging the consecutive requests from the same app. Third, we infer the geographical coordinates that user makes the network request based on the cell tower position.

In the end, we obtain our dataset contains 2,104,369 app usage records of 443 mobile users, spans 7 days from April 10, 2018 to April 16, 2018 and 14 days from May 10, 2018 to May 23, 2018. It covers 36,039 unique applications and 5,156 cell towers. Table 1 describes an example of preprocessed app usage records, including a set of fields, i.e., anonymized user identification (UserID), the launch time of the app (Time), the app that makes the network request (App), the time duration that the user has used the app (Duration) and the geographical position of cell tower (Location). By analyzing our dataset, we found the following observations.

Context-Related App Usages. The environment context has an important impact on the apps that people use. We use our dataset to investigate the relation between app usages and the environment context where people use the apps on smartphones. From Fig. 1, we can observe that people tend to use different apps in different environment context. For example, at home, people are more likely to use Game or Video apps. Social apps are used more frequently at shopping malls. We leverage the context information nearby the location of the app usage to represent the environment. We leverage the POI distributions nearby the location of the app usage to represent the environment context.

In geographic information system, a POI can be a building, a shop, a scenic spot and so on. In our work, we obtained the POI dataset of over 300,000 POIs of the city from AMap [27] (one of a leading online map provider), which provides APIs to find POIs on the map. They are classified into 23 main POI categories, including restaurants, shopping, sports, business, etc.

TABLE 1
Examples of App Usage Data

UserID	Time	App	Duration (s)	Location
1B2A7	20180510080234	Wechat	5	34.29, 109.13
5U2F1	20180510070821	Chrome	2	34.26, 109.03

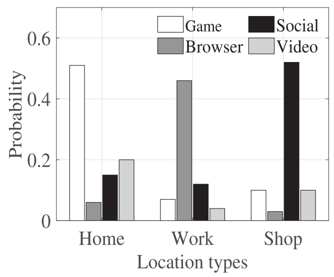


Fig. 1. The relationship between app usages and environment context.

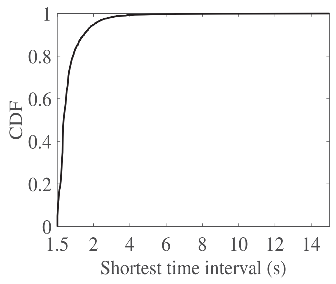


Fig. 2. CDF of the shortest time interval between the transitions of app usages of users.

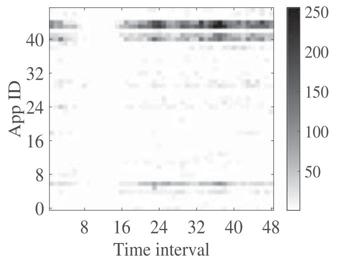


Fig. 3. The distribution of the number of app usage records of different apps in time intervals.

Real-Time App Prediction. It is critical to provide real-time app prediction for users. If a user switches frequently to different apps in a short time, the DNN agent needs to update its predicted result before launching next apps. Fig. 2 depicts the distribution of the shortest time intervals between the transitions of different app usage data of users in a day. As shown, nearly all the shortest time intervals of users are between 1 and 2 seconds. Therefore, the DNN agent is required to have the low time complexity, and thus we propose a lightweight actor-critic based personalized agent to reduce the prediction time.

Sparse App Usage Data. Adequate app usage data is also a key issue to achieve good prediction. However, it is difficult to obtain a large number of app usage data for each single user. At the beginning of the deployment, we even do not have any app usages. Fig. 3 depicts the gray value distributions of the number of app usages of different apps of a user in time intervals in a week. The result reveals that app usages are scattered over the time intervals. If we always predict those apps with higher frequency, this sometimes affects the performance. We maintain a general agent to learn the general app usage behavior of all users for personalized prediction based on the historical app usages and continuously-collected app usages of all available users.

Time-Varying App Usage Preference. Fig. 4 depicts the number of app usages of different apps that one user uses in two weeks. In the first week, she used MeiTuan a lot for online food ordering; whereas in the next week she turned to DianPing, another top online food purchase platform, maybe because DianPing provides more discount in that period. As a result, due to the time-variation of user preference on different apps, the DNN agent needs to be updated

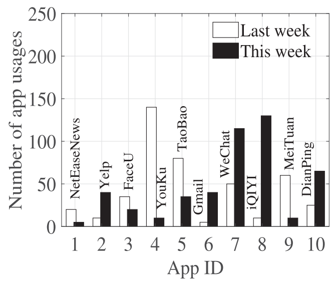


Fig. 4. The number of app usages of different apps used by a user in two weeks.

continuously. We propose an online updating strategy to learn the app usage preference incrementally.

2.3 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a promising machine learning approach, which instructs an agent to accomplish a task by trail and error in the process of interacting with the environment [28]. Four key elements are defined to describe a learning process of DRL, i.e., state, action, policy and reward.

The state $\boldsymbol{s}$ defines the input of the agent, referring to the environment representation. Different applications define different states. In DeepAPP, we define the state as the user's contextual information, including her currently-using app, surrounding environment, and current time.

The policy $\boldsymbol{\pi}$ is the core of the agent, which takes the state as input to generate an action. It learns a mapping from every possible state to an action according to the past experience. In DRL, the policy is implemented as a deep neural network (DNN).

The action $\boldsymbol{a}$ affects the environment. Every action gets a feedback from the environment. According to the feedback, we can obtain a reward $r(\boldsymbol{s}, \boldsymbol{a})$, which indicates how good or bad an action $\boldsymbol{a}$ changes the environment given a specific state $\boldsymbol{s}$. Based on the reward, a value function $Q(\boldsymbol{s}, \boldsymbol{a})$ is also defined to optimize the policy of the agent. Estimated Q value reflects the long-term effect of an action, e.g., if an action has a high Q value, the parameters of the DNN agent will be updated to favor that action. As shown in Eq. (1), $Q(\boldsymbol{s}, \boldsymbol{a})$ is the long-term reward that an agent expects to obtain in the future, where r_t is the reward of step t, and λ is the discount factor of future rewards.

$$Q(\boldsymbol{s}, \boldsymbol{a}) = E\left[\sum_{t=0}^{\infty} \lambda^t r_t | \boldsymbol{s}_0\right]. \tag{1}$$

Based on the above elements, the agent can learn to accomplish a specific task by training an agent with a specific policy, supposing we have enough transition samples $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$. The agent first perceives a state $\boldsymbol{s}$ and generates an action $\boldsymbol{a}$ by running the policy π. Then, the agent obtains a reward r given by the environment and updates the policy based on the estimate of $Q(\boldsymbol{s}, \boldsymbol{a})$. In this way, the agent and the environment interact with each other to modify the policy. After several iterations, the agent learns a stable policy. In addition, after each online inference, the agent can also use the above training process to update the policy of the DNN agent incrementally based on the new user data.

TABLE 2
Notations Used in This Paper

Notation	Description
$\mathcal{S}$ $\boldsymbol{s}$	State space, state
$\mathcal{A}$ $\boldsymbol{a}$	Action space, action
r	Reward
θ_μ, θ_Q	Parameters of the actor network and the critic network
B	App usage database
$\hat{\boldsymbol{a}}$	Proto-action
K	Number of nearest neighbors of $\hat{\boldsymbol{a}}$
$\boldsymbol{x}$	App feature
$\boldsymbol{l}$	Context feature
$\boldsymbol{t}$	Time feature
k	Prediction epoch
ω	The length of time slot
T	The combination cycle of the general agent and the personalized agent
p	The decrease rate of the balance coefficient

3 Design of DeepAPP

In this section, we introduce an overview of DeepAPP and three key techniques developed in DeepAPP. Table 2 presents the notations frequently used in this study.

3.1 Overview

DeepAPP predicts the apps that will be opened by the user in the next time slot (5 minutes in our current implementation). We perform predictions at the start of each time slot or at the moment when the user closes an app (i.e., prediction epoch). Fig. 5 depicts the architecture of our app prediction system, which consists of a back-end component and a front-end component.

3.1.1 *The Front-End Component*

It is implemented on smartphones, including two main modules, i.e., a context-sensing module and a background scheduler. The context-sensing module collects the context information (i.e., currently-using app, location and time) and sends it to the back-end component. Based on the computations on the back-end component, the predicted results are transmitted back to the front-end. The background scheduler performs a scheduling strategy to pre-load the predicted apps slightly before the next time slot.

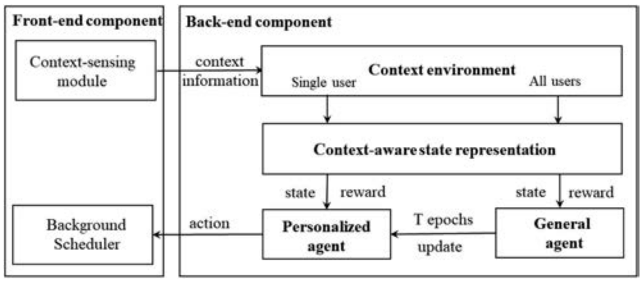


Fig. 5. The architecture of app prediction system.

3.1.2 The Back-End Component

It runs on a server and performs the training and inference for our DNN agents. It also updates the DNN agents in an online manner. The back-end component mainly consists of five modules as follows.

Context-Aware State Representation. To accurately describe a user's environment context in DeepAPP, we customize the context-aware state by a combined vector that consists of three key features, including app feature, context feature and time feature (see details in Section 3.2).

General Agent. In DRL, the agent is used to interact with the environment. The state of the environment at one moment is represented by the above environment context. The objective of an agent is to learn an optimal policy to select an action given a specific state. Since we do not have sufficient app usage data for each user, we first train a general agent using the app usage data of all users.

Action Space. Based on the perceived state, an agent predicts a set of apps that a user will open in the next time slot. In particular, the action is denoted as a binary vector $\boldsymbol{a}$, where $\boldsymbol{a}_i = 1$ indicates that app i will be opened in the next time slot. For a general agent, the action space $\mathcal{A}$ contains all feasible apps of all users. An actor-critic based agent is built to perform real-time inference in a large action space (see details in Section 3.3).

Reward Function. After the agent takes an action at the state $\boldsymbol{s}$, i.e., predicting a set of apps to a user, the user provides her feedback. She can click, or not click on these apps, and the agent receives immediate reward r according to the user's feedback, which is calculated to evaluate the prediction performance. Based on the reward, the agent updates its policy for better prediction by modifying its DNN parameters. As shown in Eq. (2), we define the reward function as the ratio between the number of correctly-predicted apps in the next time slot N_r (obtained from user feedback) and the number of predicted apps N_p (obtained from predicted result). If the number of predicted apps is 0 and the user does not use any apps in that time slot, we set the reward to 1. If the number of predicted apps is 0 or all predicted apps do not use in the predicted time slot, we set the reward to -5. Note that N_r will not be greater than N_p because the number of correctly-predicted app is always less than the number of predicted apps.

$$r = \begin{cases} 1, & N_r = 0 \wedge N_p = 0 \\ N_r/N_p, & N_r \neq 0 \wedge N_p \neq 0 \\ -5, & N_r = 0 \vee N_p = 0 \end{cases}. \tag{2}$$

Personalized Agent. During the online inference, we keep updating the general agent to a personalized agent for each user according to the real-time app usage data.

3.1.3 Two-Step Work Flow

Based on the above customized modules, DeepAPP works in two steps, i.e., the offline training and the online inference. During the offline training, we train a general agent with enough app usages of all available users. During online inference, the personalized agent is step-wise updated by optimizing the DNN parameters based on personal app usages to adapt to the time-varying app usage preference. In order to learn app usage behaviors of new apps, the general agent is also updated by app usages of all available users. The updated general agent is further used to update the personalized agent periodically (see details in Section 3.4).

3.2 Context-Aware State Representation

At each prediction epoch k, DeepAPP quantifies the context-aware state as a combined vector to represent current environment context (i.e., currently-using app, context and time) of a specific user. Specifically, the state is measured as $\boldsymbol{s}_k = [(x_k, l_k, t_k)]$, which consists of three key elements: the app feature $\boldsymbol{x_k}$, the context feature $\boldsymbol{l}_k$ and the time feature $\boldsymbol{t}_k$.

App Feature. To maintain the same dimension of input state, we construct the app feature by calculating transition times from one certain app to other apps. For a certain app i installed on the smartphone of a user, we denote the app feature of an app i at the prediction epoch k as $\boldsymbol{x}_k^i = [x_k^{i1}, x_k^{i2}, ..., x_k^{in}]$, where each $\boldsymbol{x}_k^{ij}$ is the normalized number of transition times of app i transits to app j.

Context Feature. We adopt the POI information close to a certain location to represent the context of that area. Specifically, we calculate the context feature by the distribution of POIs. For a certain location i of the user, we denote the location at the prediction epoch k as a feature $\boldsymbol{l}_k^i = [l_k^{i1}, l_k^{i2}, ..., l_k^{im}]$ for m different categories of POI (23 in our implementation). The $\boldsymbol{l}_k^{ij}$ is the number of POIs for category j within the radius of 500 meters of location i. We normalize the context feature $\boldsymbol{l}_k^i$ to represent the location at the prediction epoch k.

For the data-driven validation, we use our cellular data and quantify the context feature by the POIs around the cell tower with which the user's phone is associated. For online inference, we obtain the user's location via her smartphone and take the POIs around her location into account. By doing so, we do not need the data or any support from mobile carriers when DeepAPP is running.

Time Feature. In light of the correlation between the app usage and the usage time [29], [30], we construct the time feature as a one-hot feature $\boldsymbol{t}_k$ with the dimension of $24 * 60/\omega$, where ω is the length of each time slot (unit in minutes). It is an effective way to discretize time information. We set the time slot of current app usage to 1, and other time slots are set to 0.

In our design, new context features can be easily added to present the contextual information of users in more details, such as GPS locations [13], smartphone status [12] and Wi-Fi information [31], [32].

3.3 Actor-Critic Agent for App Prediction

Deep Q-network based methods [18], [19] have been proven to be effective for the design of the policy of the agent in case of the complex environment. In general, DQN has two main architectures. For the first architecture, it takes only the state space and outputs Q-values of all actions, which is suitable for the problem with small action space (e.g., playing Atari). The second takes the state and the action as the input of neural networks and outputs the Q-value corresponding to this action. However, this architecture needs to estimate the Q-values of all actions respectively, corresponding to $|\mathcal{A}|$ times of evaluations of the neural network,

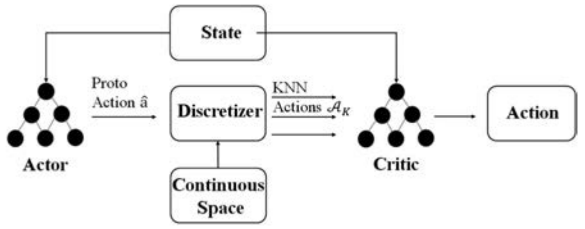


Fig. 6. The architecture of the actor-critic agent.

where $|\mathcal{A}|$ is the size of action space. This causes high time complexity ($O(n)$). Therefore, DQN-based methods cannot be used for DeepAPP, which has a large action space since users may use different combination of apps in one time slot.

Recently, some advanced techniques, such as Deterministic Policy Gradient (DPG) [33] and Deep Deterministic Policy Gradient (DDPG) [34], have been proposed to operate efficiently on the continuous space. They directly learn the mapping between the state space and the action space, and hence avoid to evaluate a large number of actions. Inspired by the above recent progresses in reinforcement learning, we propose an actor-critic based agent architecture for DeepAPP [21], [34], which operates efficiently with a large action space.

Fig. 6 depicts the design of our proposed actor-critic based architecture for the policy of both personalized and general agents. The basic idea is to allow the generalization over action space. We only need to evaluate a few actions (i.e., K in our implementation) that are close to the optimal action. By reducing the evaluation times, we minimize the computation time of one inference in DeepAPP. Specifically, the architecture includes four main components, i.e., a continuous space, an actor network, a discretizer and a critic network. We first develop the continuous space which expands from the integer action space. Then, the actor network takes the represented contextual information as input and outputs the predicted result (proto-action $\hat{\boldsymbol{a}}$) in the continuous space, which may not be in the original integer action space $\mathcal{A}$. Next, the predicted result is passed to the discretizer to find the most likely K actions set $\mathcal{A}_K$ in the action space, which are the actions close to the proto-action $\hat{\boldsymbol{a}}$. Finally, we adopt the critic network to select the optimal set of apps $\boldsymbol{a}$ with highest Q value in $\mathcal{A}_K$.

Continuous Space. The conventional action space is defined by a binary vector, in which all bits are '0' except one '1', referring to as the specific apps that people will be opened in the next time slot or not. We design a continuous space [21], which is a relaxed version of the original action space, which achieves generalization over actions. It maps similar actions into a close adjacent space. We then can find an approximate solution and evaluate adjacent actions around it to obtain the optimal predicted results. According to our customized design of action space for app prediction, we can easily achieve this. Specifically, we expand the action space to a continuous space, which is defined in the real field rather than the integer field. Each item in the vector can be a real number between 0 and 1.

Actor Network $\mu^{\boldsymbol{\theta}}$. We design the actor network as $\mu^{\boldsymbol{\theta}}(s)$ that maps from the context-aware state space $\mathcal{S}$ to the action space $\mathcal{A}$, where $\mu^{\boldsymbol{\theta}}$ is the mapping function defined by parameters $\boldsymbol{\theta}_\mu$. Given the state $\boldsymbol{s}$ of the environment, the actor network directly outputs an approximate predicted result, denoted as proto-action $\hat{\boldsymbol{a}}$. By evaluating the actions around $\hat{\boldsymbol{a}}$, we can avoid to search in the whole action space, and thus reduce the prediction time of our system. However, proto-action $\hat{\boldsymbol{a}}$ may not be in the action space $\mathcal{A}$. Therefore, we use a discretizer to map from $\hat{\boldsymbol{a}}$ to an action $\boldsymbol{a} \in \mathcal{A}$.

Discretizer. Normally, the actions with lower Q values may occasionally fall near the proto-action $\hat{\boldsymbol{a}}$, which causes errors in the predicted result. Additionally, some actions close in the action space may have different long-term Q values. In the context of these circumstances, it is not advisable to simply select the closest action to $\hat{\boldsymbol{a}}$ as the final result. To avoid selecting an outlier action, we develop a discretizer, which maps from the continuous space to a set of adjacent actions $\mathcal{A}_K$. As shown in Eq. (3), we enumerate all actions in $\mathcal{A}$ to find K actions $\mathcal{A}_K$ that are close to the proto-action $\hat{\boldsymbol{a}}$.

$$\begin{aligned} &min_{\boldsymbol{a}\in\mathcal{A}}||\boldsymbol{a} - \hat{\boldsymbol{a}}||_2 \\ &s.t. : a_i \in \{0,1\}, \quad a_i \in A_u, \end{aligned} \tag{3}$$

where A_u is the set of apps on a user's smartphone. With Eq. (3), we use euclidean distance to estimate an action instead of evaluating the value function in the form of neural network. This leads to much low time complexity for one estimation. Moreover, we also use a nearest neighbor algorithm [35] to reduce the overall evaluation times, which takes the time complexity of $O(log(n)$ to find the best action.

Critic Network $Q^{\boldsymbol{\theta}}$. We finally adopt a critic network to find the action in $\mathcal{A}_K$ with the maximum Q as our result. The critic network is the other neural network $Q^{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$, which returns the Q value for each action $\boldsymbol{a} \in \mathcal{A}_K$, like DQN. Compared with the DQN-based methods that evaluate all actions in $\mathcal{A}$ to find a proper action, we only evaluate a few actions in $\mathcal{A}_K$. Specifically, the critic network takes each action $\boldsymbol{a}$ in $\mathcal{A}_K$ and the state as input to find the action with the largest value function $Q(\boldsymbol{s}, \boldsymbol{a})$ as the final prediction result, as presented in Eq. (4).

$$Q(\boldsymbol{s}, \boldsymbol{a}) = argmax_{\boldsymbol{a}\in\mathcal{A}_K} Q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta_Q}). \tag{4}$$

3.4 Online Updating of the Agents

Due to the data sparsity problem, we train a general agent with the app usage data of all users at the beginning. We then use the general agent to perform app prediction for each individual user and gradually update it to a personalized agent using the personal app usage data of each user. As app usage data are collected from all available users, we also update the general agent by newly-collected data periodically, and further use the periodically-updated general agent to enhance the personalized agent.

3.4.1 *Offline Training of the General Agent*

During the offline training, DeepAPP trains a unified general agent with the app usage data of all users. Then, DeepAPP leverages the pre-trained model for the initialization for online inference of each user. To ensure that the general agent can be used for the personalized agent, we maintain the same structure for these two agents. Their DNN networks have the same network topology with the same input and output. For input, we transform the feature of the contextual environment into a fixed length vector to represent the state. Regarding with output, the length of the output dimension is the same for all users' models, i.e., the action space is composed of all the possible apps of all users.

3.4.2 Online Updating of the General Agent

During online inference, we update the general agent at regular time intervals (one day in our implementation). The online updating of DeepAPP normally has two steps, i.e., the update of critic network and the update of actor network.

First, the agent optimizes the loss function L to update the critic network based on the long-term expected reward Q, which could be estimated by the interaction between the user and apps. The loss function L is defined as Eq. (5)

$$L = \frac{1}{N}\sum_{i=1}^{N}(Q_{tgt} - Q(\boldsymbol{s}_i, \boldsymbol{a}_i|\boldsymbol{\theta}_Q))^2, \tag{5}$$

where N is the number of app usage transitions from all users and the frozen Q value Q_{tgt} is learned by the target networks [18] as shown in Eq. (6).

$$Q_{tgt} = r_i + \lambda Q(s_{i+1}, \mu\prime(s_{i+1}|\boldsymbol{\theta}_{\mu'})|\boldsymbol{\theta}_Q), \tag{6}$$

where λ is the discount factor. With back propagation, the critic network can be easily updated according to the gradient of loss function $\nabla_{\boldsymbol{\theta}_Q} L$.

We further update the actor network of the agent using Eq. (7)

$$\nabla_{\boldsymbol{\theta}_\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}_Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\boldsymbol{\theta}_\mu} \mu(\boldsymbol{s}|\boldsymbol{\theta}_\mu)|s_i, \tag{7}$$

where $\mu(s|\boldsymbol{\theta}_\mu)$ is the K predicted results of the actor network and $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}_Q)$ is the evaluations of the K actions calculated by the critic network with Eq. (4). The gradient in Eq. (7) is calculated by the derivative rules of compound functions to optimize the parameters of actor network.

3.4.3 Online Updating of the Personalized Agent

The personalized agent has the same updating algorithm as the general agent, as specified in Eqs. (5), (6), and (7). The only difference is the updating frequency. We update the personalized agent more frequently at the start of each time slot (5 minutes in our current implementation) or when the user closes one app. High updating frequency makes the personalized agent rapidly adapt to the time-varying app usage preference.

3.4.4 Combination of the Personalized Agent and the General Agent

We combine the parameters of the actor network $\boldsymbol{\theta}_\mu$ in the general agent and the parameters of the actor network $\boldsymbol{\theta}_{\mu_g}$ in the personalized agent as Eq. (8).

$$\boldsymbol{\theta}_\mu = \boldsymbol{\theta}_\mu + \eta\boldsymbol{\theta}_{\mu_g}, \tag{8}$$

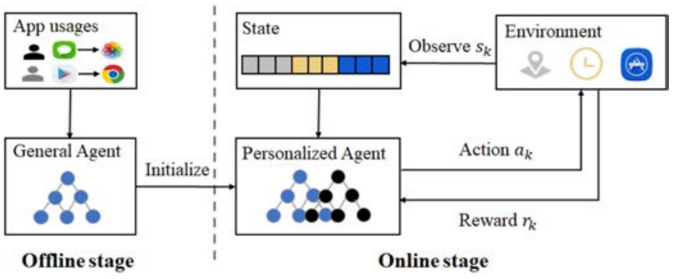


Fig. 7. The work flow of DeepAPP.

where a balance coefficient η is adopted to adjust the importance between the general agent and the personalized agent. As time goes on, we could obtain more individual app usage data and make good prediction based on the personalized agent. If we still keep a large weight of the general agent, the general app usage behaviors learned by the general agent may overwhelm the personal app usage behavior of the personalized agent. Therefore, with the increase of prediction epochs, we decrease η linearly with a decrease rate of p.

3.5 Work Flow of DeepAPP

We present the work flow of DeepAPP in Fig. 7, which mainly consists of two stages: offline training stage and online stage. We first randomly initialize the parameters $\boldsymbol{\theta}_\mu$ of the actor network $\mu^{\boldsymbol{\theta}}$ and the parameters $\boldsymbol{\theta}_Q$ of the critic network $Q^{\boldsymbol{\theta}}$, which denoted as main networks. If we directly evaluate the actor and critic networks to obtain the Q_{tgt} value in Eq. (5), the training process will be unstable [36]. To address this problem, we create the copies of actor and critic networks ($\mu'^{\boldsymbol{\theta}}$ and $Q'^{\boldsymbol{\theta}}$) [18], which denoted as target networks. The target networks have the same initial parameters with the main networks. The target networks are updated slower (every 100 prediction epochs in our implementation) than those of the main networks. The parameters of the updated target networks will be used to calibrate the main networks during online stage later. At the same time, the networks represented by DNNs require the input data to be independent and identically distributed (i.i.d), but the training data (app usages) are highly related, leading to training instability. Hence, we also develop an app usage database B [18] to break the correlation between the app usage sequences.

Offline Stage. As depicted in the left part of Fig. 7, we leverage the offline app usage data from all the available users in the app usage database B to train the networks of the general agent. We first convert all users' app usage data into transition samples ($<\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1}>$) and store them into B. By randomly sample N app usage transitions from B, we optimize the network parameters as defined in Eqs. (5) and (7).

Online Stage. As the right part of Fig. 7 depicts, the personalized agent performs prediction and updates its network for better prediction accuracy based on the initial trained general agent. At each prediction epoch k, DeepAPP first obtains the context-aware state $\boldsymbol{s_k}$ from the environment sensed by the front-end component as the input of the personalized agent and derives a proto-action $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$ by the actor network μ^{θ}. In order to explore potential better actions, we introduce a stochastic exploring mechanism by adding random noise into the action. Specifically, we add a random noise $\epsilon \boldsymbol{I}$ to the proto-action $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$ [34], which has the similar idea as ϵ-greedy [37]. As ϵ decreases with the prediction epoch, more certain action will be taken with more training. $\boldsymbol{I}$ is a homotypic vector with action, which follows the standard uniform distribution $(U(0,1))$.

Then, we find K nearest actions of the proto-action $\hat{\boldsymbol{a}}_{\boldsymbol{k}}$ by solving Eq. (3). The possible actions are passed to the critic network for evaluating the Q-value of each action. The action with highest Q-value is selected and passed to background controller at the front-end component of a specific user. According to the feedback from the user, the personalized agent calculates the reward r_k to update the actor and critic networks.

As introduced in the initialization stage, in order to obtain a stable estimate of Q_{tgt} value, we update the target networks every 100 prediction epochs. In order to limit the updating speed of the target networks, we adopt soft update technique [18] to stabilize the parameters of the target networks.

$$\begin{aligned}\boldsymbol{\theta}_{Q'} &:= \tau\boldsymbol{\theta}_{Q} + (1-\tau)\boldsymbol{\theta}_{Q} \\ \boldsymbol{\theta}_{\mu'} &:= \tau\boldsymbol{\theta}_{\boldsymbol{\mu}} + (1-\tau)\boldsymbol{\theta}_{\mu}.\end{aligned} \tag{9}$$

Finally, at every T combination cycle, we update the general agent and then combine it with the personalized agent as Eq. (8).

4 Implementation

In this section, we introduce implementation details of the back-end component and the front-end component respectively. The source code of DeepAPP is available on the website [38].

4.1 Back-end Component

At the back-end side, DeepAPP trains a unified general agent for all users, and performs inference and update of the personalized agent for each user. All agents are implemented on TensorFlow [24] and share with the same network structure. They use two 2-layer fully-connected feedforward neural networks for the actor network and the critic network. This leads to the total number of parameters of our model $H_1|S| + H_1|A| + H_2|A| + B|S| + B|A| + H_1H_2$, corresponding to the space complexity of $O(n^2)$. Among which, $|S|$ is the dimension of input vector, $|A|$ is the dimension of output vector, H_1 and H_2 are the number of neurons of 2-layer fully-connected feedforward neural network, B is the maximum size of app usage dataset. In our implementation, we set $H_1 = 1000$ and $H_2 = 1000$ for the actor network, and $H_1 = 400$ and $H_2 = 200$ for the critic network.

We adopt $ReLu$ activation function for the fully-connected neural networks, and $Tanh$ activation function for the output layer. To alleviate the over-fitting problem, we introduce an L_2 regularization term in the loss function [39], besides the regularization coefficient of loss of actor network is different from critic network's. Besides, there are a few hyper-parameters to set in both networks. We conducted a comprehensive empirical study to find best settings, as shown in Table 3.

TABLE 3
Hyper-Parameter Settings

Hyper-parameter	Setting
Batch size N	32
Number of the offline training iterations	500,000
Future reward discount λ	0.90
The maximum size of app usage database $\boldsymbol{B}$	100,000
Learning rate of actor network $\nabla\mu$	0.0001
Learning rate of critic network ∇Q	0.001
Soft updating coefficient τ	0.01

The back-end component is implemented on a server, which contains 2 CPUs. Both CPUs have dual Intel(R) Xeon(R) CPU E5-2609 v4 @ 1.70 GHz with 8 cores. Experiments demonstrate that a 2-core CPU is enough to support to make an inference within 0.31 second. We also use a GPU cluster with 2 nodes (12 GB memory) to accelerate the offline training of the general agent, which can save 2.47× training time compared with the training on CPUs.

4.2 Front-End Component

The front-end is implemented as a customized app on smartphones (our current implementation runs on Android 9.0). The implementation of the app includes two modules, i.e., a context-sensing module and a background scheduler.

Context-Sensing Module. We use the context-sensing module to obtain the real-time context information of users (e.g., user feedback, location, time and currently-using app) for the next time prediction. Note that all sensitive information are acquired on a voluntary basis. Specifically, we obtain the location information through *LocationManager* provided in Android SDK, and the current foreground app through *AccessibilityEventEvents* by accessibility services and smartphone status through Android logcat.

Background Scheduler. We only pre-load the apps that may be used next to minimize the energy and memory cost of pre-loading apps on smartphones. To do so, we develop a background scheduler, which pre-loads the apps before next time slot according to the predicted results. In particular, we use *getLaunchIntentForPackage* in Android *PackageManager* to realize the pre-loading.

4.3 Data Transmission

The data transmitted between the back-end component and front-end component are small in size. The data transmission can be supported either by Wi-Fi or cellular networks. First, we need to transmit the context-sensing result from the front-end component to the back-end component. In each iteration of prediction, we only need to send about 480 bytes on average, including user feedback, location, time and currently-using app. The transmission delay is less than 30 ms on average if cellular networks are used. Since Wi-Fi is faster than cellular networks, the transmission delay can be further reduced if Wi-Fi networks are available. At the same time, we need to transmit the predicted

TABLE 4
Parameter Settings

Parameter	Setting
The length of time slot ω	5 minutes
The number of nearest neighbors of proto-action K	5% of $\lvert\mathcal{A}\rvert$
The combination cycle T	one day
The decrease rate of balance coefficient p	0.09

result from the back-end component to the front-end component. The predicted result is composed of a string of app IDs. The information can be encapsulated in one packet within 120 bytes. The transmission delay is 25 ms on average.

4.4 Inference on Smartphones

DeepAPP can support making inference on the smartphone of each user, without the need of a back-end component. This improves the scalability of DeepAPP. We use TensorFlow Lite [25] as a solution to run DeepAPP on smartphones. TensorFlow Lite is a widely-used developing tool to deploy machine learning models on mobile devices with low latency and memory cost. To implement the DNN agent on the smartphone, three operations have to be performed, i.e., training a unified general agent, converting the agent to a TensorFlow Lite format, and integrating the model in the app. We first use all users' app usage data to train a DNN agent. We export the DNN agent to a *tf.GraphDef* file by the interface (*tensorflow.gfile.FastGFile*) provided by TensorFlow Lite. It ensures that the agent model can communicate with our app. Finally, we integrate the DNN agent into our customized app.

Since TensorFlow Lite currently does not support training operation on mobile devices [40], we only run DeepAPP for inference, but do not update the personalized agent on smartphones. If a user chooses to run DeepAPP locally, she will not need to transmit her data to the back-end component. However, without further updating, the system performance may decline over time.

5 Data-Driven Evaluation

We first conduct data-driven evaluations of DeepAPP on the dataset introduced in Section 2.2. It includes the app usage data from a period of 21 days. The parameter settings of DeepAPP are shown in Table 4. We use these settings by default in the following experiments. In Section 5.4, we explain how we set these parameters to the best values.

5.1 Experiment Settings

Training and Testing. We divide the dataset into two parts, i.e., 14-day data for training and 7-day data for validation. Cross-validations, by repeating the experiments with different partitions of the training and validation data, have been conducted.

Performance Criteria. We use precision and recall to measure the app prediction accuracy. Precision is defined as the average ratio between the number of correctly-predicted apps and the number of all predicted apps in the next time slot. Recall is the average ratio between the number of correctly predicted apps and the number of real used apps of all users in the next time slot. In addition, we also measure the average execution time to measure the efficiency.

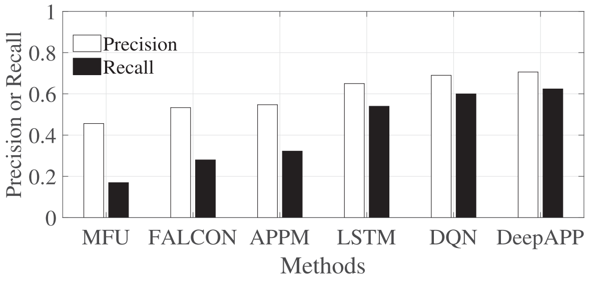


Fig. 8. Prediction accuracy.

Benchmarks. We compare the prediction accuracy of DeepAPP with following 5 baselines. All the parameters of baselines are set to the optimal values according to the empirical experiments on our dataset.

- *MFU.* Intuitively, we can always predict the next apps as the most frequently used (MFU) M apps, which can be found based on the number of app usage records of each user in our dataset. M is set to 5 in our implementation.
- *FALCON.* Yan *et al.* [1] develop an effective context-aware app prediction model by utilizing context features (i.e., app, location and time). In view of the special localization method of our dataset (cellular-based localization), we derive the location information ("Home", "Work place" and "On the way") by the method introduced in [41]. In that work, the authors proposed a cluster-based technique for analyzing cellular-based dataset to identify important locations for users.
- *APPM.* Parate *et al.* [4] first leverage Prediction by Partial Match (PPM) model [42] to predict next apps and use the distribution of the usage time of the apps to infer the app usage time. We keep the apps in the next time slot as the predicted results.
- *LSTM.* Xu *et al.* [16] formulate the app prediction problem as a multi-label classification problem and propose a LSTM-based model to predict the most likely apps. We select the apps with the probabilities higher than 0.8 as the predicted results. We also incorporate our context-aware state into their model.
- *DQN.* We also implement a DQN-based app prediction scheme [18]. It is an inefficient way to leverage deep reinforcement learning in app prediction. We also implement our other designs, like the context-aware state and online updating, in this DQN-based scheme.

5.2 Overall Performance

We first present the prediction accuracy of DeepAPP on our dataset.

5.2.1 Prediction Accuracy

Fig. 8 depicts the average prediction accuracy on the validation data. From the experiment result, we can see that

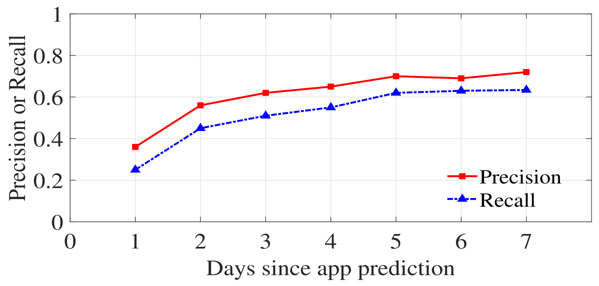


Fig. 9. Evolution of prediction accuracy over time.

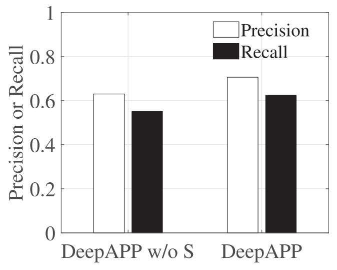


Fig. 10. Context-aware state.

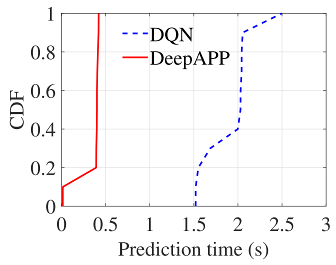


Fig. 11. Prediction time.

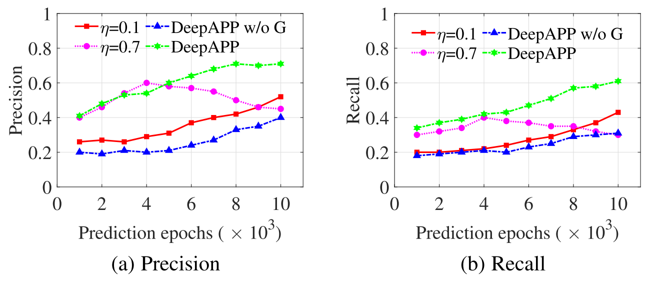


Fig. 12. General agent.

DeepAPP provides the best prediction accuracy among other baselines. The reasons are as follows. First, DeepAPP learns a data-driven model-free agent to make prediction rather than traditional explicit models (Markov chains and Bayesian framework). The data-driven policy of the prediction agent can take complex environment context as input. Second, with reinforcement learning, DeepAPP can learn to explore the large prediction space automatically and effectively and select the best set of apps for the prediction result simultaneously, while other methods can only predict the apps with highest probabilities separately. This ignores relationship among the predicted apps, which is unreasonable in the real scenario. We also find that DeepAPP has similar performance as the DQN-based scheme. The specific design of the DNN agent of DeepAPP focuses on reducing the time complexity of make an inference, which will be studied in Section 5.3.2.

5.2.2 *Evolution of Prediction Accuracy Over Time*

Fig. 9 presents the precision and recall of all mobile users on each day during a 7-day test. The prediction performance improves over time, which means DeepAPP can adapt well to app usage dynamics by updating the personalized agent online. The result also confirms the effectiveness of online update strategy in solving the time-varying prediction problem, allowing rapid adaptation to the change of app usage preference.

5.3 Effectiveness of the Proposed Modules

In this subsection, we evaluate the effectiveness of the proposed three modules in DeepAPP.

5.3.1 *Performance Gain of the Context-Aware State*

To verify the effectiveness of our context-aware state representation, we implement another version of DeepAPP (denoted as "DeepAPP w/o S") by only vectorizing the semantic locations (i.e., "Home", "Work place" and "On the way"). As depicted in Fig. 10, the precision and recall of DeepAPP is 7.6 and 7.3 percent higher than those of DeepAPP w/o S. This is because DeepAPP w/o S cannot learn the app usage behaviors at some contexts where users have not been to or do not have semantic information.

5.3.2 *Fast Prediction Time of Actor-Critic Agent*

DeepAPP has the ability of fast inference compared with the basic DQN model. We use app usage data of all users to test the average prediction time of two DRL-based methods (i.e., DQN and DeepAPP). Fig. 11 depicts the CDF of the prediction time. The average prediction time of DeepAPP (0.31 seconds) is far less than that of DQN (2.04 seconds). This indicates that our lightweight actor-critic based agent can effectively reduce the prediction time and enable real-time app prediction.

5.3.3 *Performance Gain of the General Agent*

We verify the effectiveness of the general agent in DeepAPP. We implement two versions of DeepAPP. The first version discards the general agent, which denoted as "DeepAPP w/o G". The second version adopts two fixed balance coefficient η values to combine the personalized agent with general agent while online inference, which denoted as "$\eta = 0.1$" and "$\eta = 0.7$".

Fig. 12 depicts prediction details of these three methods during online learning within 10,000 prediction epochs. With the increase of epochs, both the precision and recall increase. DeepAPP is consistently higher than the DeepAPP w/o G during online learning. The results confirm that the general agent succeeds in solving the data sparseness problem.

Besides, we find that the performance of DeepAPP with a linearly decreasing η value is superior to that under a fixed η value. For instance, under a larger η (i.e., 0.7), DeepAPP works well at the beginning, but weakens at the later stage. With a small η (0.1) and vice versa. A

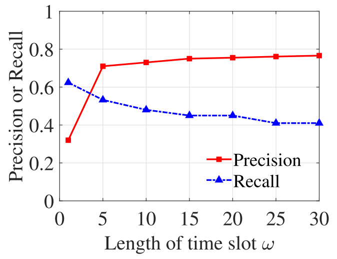


Fig. 13. Effect of the length of time slot ω.

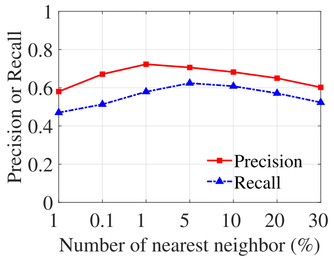


Fig. 14. Effect of number of nearest neighbors K.

TABLE 5
Ratio of the Training Data to the Validation Data

Ratio	9:12	11:10	14:7	15:6	18:3
Precision	61.9%	67.3%	70.6%	69.2%	67.8%
Recall	52.6%	59.1%	62.4%	61.7%	59.0%

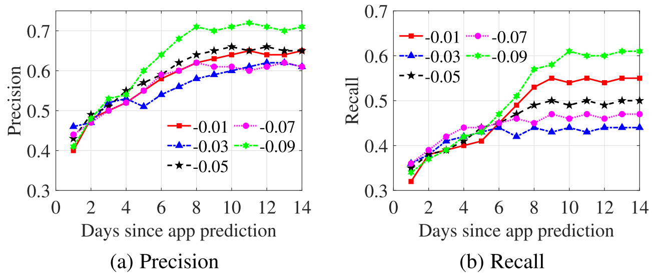


Fig. 15. Effect of the decrease rate p.

linearly decreasing η value can always maintain high precision and recall over time. We adopt a bigger η at the beginning, which addresses the data sparsity problem. As prediction epochs increase, we reduce the role of the general agent and let the personalized agent dominated by the individual app usage data.

5.4 Parameter Settings

We test the choices of four parameters in DeepAPP, i.e., the ratio of the training data to the validation data, the length of time slot ω, the number of nearest neighbors of proto-action K and the decrease rate of the balance coefficient p.

Ratio of the Training Data to the Validation Data. Table 5 presents the variation of accuracy as the ratio of the training data to the validation data varies. We discover that the increase of 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. Therefore, we select the ratio at 14:7 with best performance.

Length of Time Slot ω. Fig. 13 depicts the performance of DeepAPP by varying the length of time slot ω from 1 minute to 30 minutes. As ω increases, the precision increases, but the recall gradually decreases. We select a proper ω using F-Score [43], which achieves a balance between the precision and recall. As depicted in Table 6, we can find the F-Score reaches its maximum at ω =5. This also means that in real scenarios, a 5-minute time slot should be recommended.

The Number of Nearest Neighbors K. The motivation of the number of nearest neighbors of proto-action is to lower the impact of noisy actions which may occasionally fall near the proto-action. We conduct an experiment to select a proper K. Fig. 14 shows the variation of accuracy by varying the number of nearest neighbors K from 1 to 30 percent of $|\mathcal{A}|$. As depicted, when $K = 1$, the accuracy is worst, which proves the rationality of selecting K nearest neighbor to find the optimal action. When $K > 1$, the technique can filter out noise actions which occasionally fall near the proto-action, resulting in the enhancement of the precision and recall of DeepAPP. As shown in Table 7, we also select a default $K = 5\%$ of $|\mathcal{A}|$ using F-Score [43] as the default setting in the experiments.

The Decrease Rate p. In our design, we adopt an adaptive balance coefficient, which gradually reduces the weight of the general agent in the update of each personalized agent. We evaluate the performance of DeepAPP with different decrease rate of the balance coefficient from 0.09 to 0.01. Fig. 15 depicts the performance of DeepAPP under various values of decrease rate with respect to the prediction epochs. Our method can maintain high precision and recall when p is set to the largest. This indicates that a larger decrease rate p should be considered in the real application scenario.

5.5 Robustness

In this section, we evaluate the system robustness of DeepAPP according to different attributes of training data, i.e., the number of dominant apps, the number of installed apps and the number of app usage records.

Impact of the Number of Dominant Apps. The dominant app refers to the most frequent apps of an individual. If a

TABLE 6
The F-Score With Different Length of Time Slot ω

ω (min)	1	5	10	15	20	25	30
F-Score	0.584	0.589	0.579	0.563	0.564	0.532	0.534

TABLE 7
The F-Score With Different K

K (x% of $\lvert\mathcal{A}\rvert$)	1	0.1%	1%	5%	10%	20%	30%
F-Score	0.519	0.581	0.643	0.662	0.643	0.608	0.560

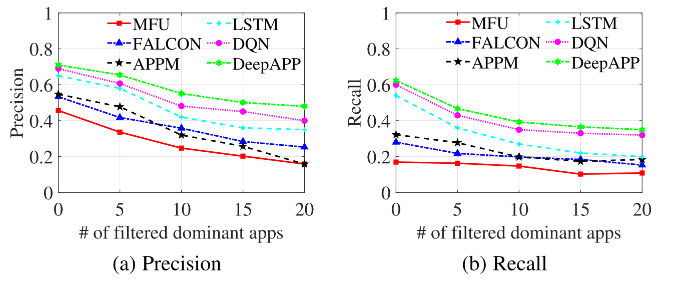


Fig. 16. Effect of dominant apps.

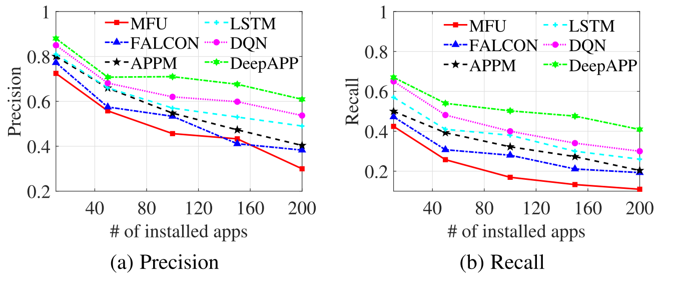


Fig. 17. Effect of the number of installed apps.

user only uses several apps frequently, the prediction model can always predict these apps each time, making it easier to predict. On the contrary, if a user involves cross-use of various types of apps, it will be difficult to predict. In this experiment, we study the impact of number of dominant apps on the accuracy of app prediction. We filter out the app usage records that are triggered by the dominant apps.

As depicted in Fig. 16, the different approaches have a severe performance degradation in precision and recall as the filtered number of dominant apps increases. For example, when the number equals to 10, the app prediction precision is about 55.1 percent and the recall is 39.2 percent. This indicates that DeepAPP works effective in the scenarios of low diversity of app usages.

Impact of the Number of Installed Apps. We also investigate the impact of the number of installed apps. When a user installs a large number of apps on smartphones, it will be more difficult to predict the next apps. We categorize the number of installed apps into 5 levels. Fig. 17 depicts the experiment results. As shown, the precision and recall decrease sharply as the number of installed apps increases. Especially, when the number of installed apps is less than 10, the precision and recall are reached 88 and 58 percent, respectively. For a larger value ($\geq$200), the precision and recall are only 60.1 and 40.9 percent. The experiment results demonstrate that DeepAPP works better in the case of fewer installed apps on smartphones.

Impact of the Number of App Usage Records. The number of app usage records that are used for training the model may have impacts on the performance. We explore how DeepAPP performs when the number of app usage record is different. We categorize the number of app usage records into 5 levels, i.e., {< 50}, {≥ 50 & < 100}, {≥ 100 & <200}, {≥200 & <400} and {≥400}. Fig. 18 depicts the performance on different number of app usage records.

As shown, with the increase of app usage records, the precision and recall are also enhanced. For example, as the

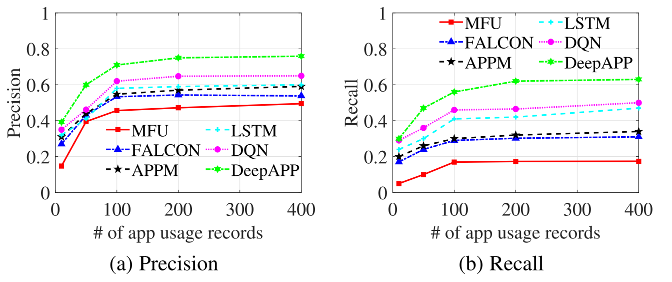


Fig. 18. Effect of the number of app usage records.

number of app usage records increases from 50 to 200, both the precision and recall values increase sharply (e.g., 71.0 percent in precision and 56.0 percent in recall). This indicates the minimum training samples for our system to achieve an acceptable accuracy is about 200.

6 Field Study

We also test DeepAPP by field experiments from 17 Sep. to 10 Nov. 2018. Compared with data-driven evaluations, in the field experiment, we can not only measure the accuracy of DeepAPP, but also collect the real user experience on DeepAPP. We recruit 29 participants and collect app usage records as ground truth. Participants include 13 females and 16 males, aged from 19 to 49, which have various occupations such as company employees, college teachers and students, etc. After participants agree to take part in the experiment, we install the Android application introduced in Section 4.2 on smartphones and monitor their app usage traces. We also collect their smartphone status such as power consumption and memory usage for the analysis of system overhead. At last, all participants successfully completed the experiment, and in all we collected 76,021 pieces of app usage records during the 55-day field experiment.

As mentioned in Section 4.4, the mobile devices can only support making inference of the neural network model currently, not training and updating. To this end, we deploy a system as the architecture in Fig. 5. In this way, the participants upload their app usage data to our server. We use the app usage data of all available users to train a general agent as the initial prediction model for each subject. During the field study, we use the personal data of each subject to update each subject's model on the server side for better prediction.

6.1 User Survey

We ask participants to complete a weekly questionnaires to collect the feedback on the usability of DeepAPP. Questionnaires are designed in a Likert scale format [44], which require participants to rate a statement from "strongly disagree (1)" to "strongly agree (5)". The results show that 87.51 percent of users are satisfied with our app prediction system, which is an alternative proof that our predictive model is effective and 71.88 percent of participants agree that the app can save their time of launching apps by preloading our predicted apps into the memory.

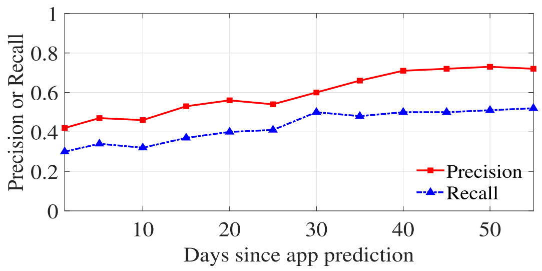


Fig. 19. Accuracy during the field experiment.

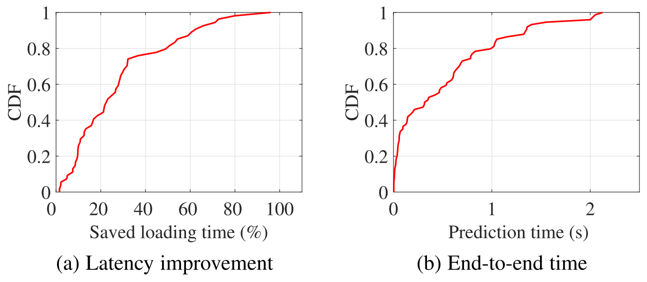


Fig. 20. Performance analysis.

6.2 Performance Analysis

We analyze the performance of our field experiment from 3 aspects, i.e., accuracy, latency improvement and end-to-end prediction time.

Accuracy. Fig. 19 depicts the evolution of precision and recall over time during the field experiment of DeepAPP. As expected, like the data-driven evaluations, DeepAPP can also quickly adapt to the time-variation of user preference and achieve stable accuracy over time.

Latency Improvement. We use the average ratio of the saved loading time to the launch time of smartphones without deploying DeepAPP to evaluate the time reduction on participants' smartphones. We profile the launch time of all installed apps on participants' smartphones. Then, we could obtain the time reduction according to the correctly-predicted result of the participants. This measurement ignores the launch time of apps if DeepAPP has pre-loaded the apps, which is neglectable in practice [2]. Fig. 20a shows that our system can reduce the app loading time by 68.14 percent on average compared with no pre-loading.

End-to-End Prediction Time. The end-to-end prediction time is very important and directly related to user experience. We calculate the end-to-end prediction delay by the time difference between the start time of uploading the context information and the end time of receiving the predicted result, which can be easily obtained by the Android logcat from participants' smartphones. From Fig. 20b, we can see that prediction delay is negligible, i.e., less than 1 seconds of 80 percent, including both prediction computation and data transmission between the back-end component and the front-end component.

6.3 System Overhead

DeepAPP may produce two types of overhead, i.e., 1) the power consumption and memory cost of running DeepAPP prediction and 2) the power consumption and memory cost caused by the apps pre-loaded by DeepAPP. Because it's difficult to directly measure the power consumption from Android, we measure the actual power consumption of apps by using a widely-used power monitoring application (AccuBattery [45]) in the existing works [46], [47], which supports to obtain the power consumption using the battery management system in smartphones since Android 5.0.

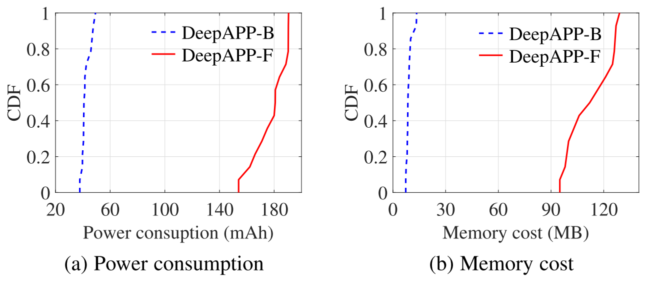


Fig. 21. System overhead of DeepAPP prediction.

6.3.1 Overhead of DeepAPP Prediction

We test the overhead of DeepAPP prediction on 2 participants with the same model of smartphones (Honor 20 Pro) on Android 9.0. We implement two versions of DeepAPP of running app prediction, i.e., making inference on the back-end server (denoted as DeepAPP-B) and making inference on the front-end (denoted as DeepAPP-F).

Power Consumption. As depicted in Fig. 21a, the extra cost of DeepAPP-B and DeepAPP-F are about 42.48 mAh and 178.87 mAh on average in a day. Compared with DeepAPP-F, DeepAPP-B has less power consumption. This is because DeepAPP-B performs prediction inference and agent updating at the back-end server, saving the power consumption of smartphones. The customized design of the context-aware module in DeepAPP does not cause additional power consumption, compared with other systems [1].

Memory Cost. Fig. 21b depicts that the memory cost and computation requirement of two versions of DeepAPP. The results reveal that the average memory cost of DeepAPP-B is less than 9.3 MB, and does not consume much extra memory (i.e., 113.6 MB) during making inference on the front-end. Current smartphones, like Samsung Galaxy S9 and HUAWEI Mate 10 Pro, have at least 4 GB memory and 8-core CPU, which can totally support DeepAPP online inference without a back-end support.

6.3.2 Overhead of App Pre-Loading

The overhead of app pre-loading is in two aspects: power and memory. We also test the overhead of pre-loading on 4 participants with the same model of smartphones (HUAWEI Mate 10 Pro).

Power Consumption. As apps share hardware components, loading apps simultaneously will save more power than loading apps separately [48], and thus the power consumptions of users are less than what we estimate. Fig. 22a depicts the estimated average power consumption in different days. We find that the app consumes less than 2.18 percent of battery powers of participants' smartphones on average in a day. The reasons are as follows. First, DeepAPP does not pre-load unpredictable apps, which will not

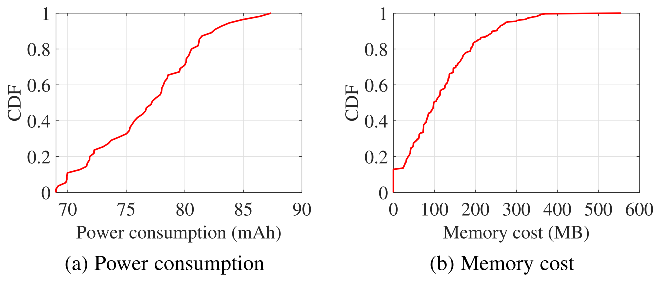


Fig. 22. System overhead of app-preloading.

consume any additional power consumption. Second, DeepAPP only introduces the few additional power consumption by misprediction due to the high precision of DeepAPP.

Memory Cost. Due to app pre-loading will bring extra memory cost of smartphones, we further test the memory usage on users' smartphones. With the users' consent, we monitor the memory usage of participants and obtain a result in Fig. 22b. As shown, app pre-loading does not consume much memory on average, i.e., 190.6 MB of total memory, because the background scheduler only pre-loads apps that will be used in the next time slot. Besides, if the user does not use the predicted apps, we will immediately unload the apps in memory.

7 Related Work

App Prediction. Many app prediction methods [1], [8], [9], [10], [11], [12], [13], [14], [15], [16], [49] have been designed for personalized app prediction. Huang *et al.* [11] model the app usage transition by a first-order Markov model and use the contextual information, such as time, location and the latest used app. Natarajan *et al.* [9] model the app usage sequences using a cluster-level Markov model, which segments app usage behaviors cross multiple users into a number of clusters. Bayesian framework [11] improves the performance of app prediction by combining different features. PTAN [14] combines various explicit features such as location semantics (either home or work) and implicit features such as app usage information. Parate *et al.* [4] and Zhu *et al.* [10] transform the place into semantic location to improve the performance of app prediction. Chen *et al.* [15] consider rich context by graph embedding techniques for personalized prediction. AppUsage2Vec [50] considers app attention, user personalized characteristics in app usage and temporal information for app prediction and propose a generic model to address the cold-start problem, which the model does not have sufficient data to make reliable prediction [22]. APPM [4] separately considers the prediction of a few specific apps with their launch time to prefetch in time on smartphone. However, most of them only predict the next app, without considering the launch time.

There are also some works that are orthogonal to our work. They benefit practical apps on smartphones from different perspectives. SmartIO [3] reduces the application loading delay by assigning priorities to reads and writes. HUSH [51] unloads background apps for energy saving automatically. CAS [7] develops a context-aware application scheduling system that unloads and pre-loads background applications in a timely manner. ShuffleDog [52] builds a resource manager to efficiently schedule system resources for reducing the user-perceived latency of apps.

Deep Reinforcement Learning. Mnih *et al.* solve the problem of stability and convergence in high-dimensional data input using Deep Q Network (DQN) [18]. Many technologies have been proposed to improve the performance of DQN. Double Q-learning [19] is put forward to handle overestimations of action values. Wang *et al.* [53] develop a dueling DQN architecture that presents state values and action advantages separately to promote the generalization of different actions. Previous works have further extended deep reinforcement learning to continuous action space and large discrete action space. An actor-critic based on the policy gradient [34] is presented to solve the continuous control problem. Mnih *et al.* [54] propose asynchronous gradient descent for optimization of deep neural network and show successful applications on various domains. Arnold *et al.* [21] present an actor-critic architecture which can act in a large discrete action space efficiently. Based on this architecture, our work designs a new actor-critic based agent for app prediction.

Recently, deep reinforcement learning has been studied and applied in many domains [55], [56], [57], [58], [59], [60], [61]. DSDPS [56] applies DRL for the distributed stream data processing system based on the experience rather than solving the complicated model. AuTO [57] leverages a two-tier DRL model based on the long-tail distribution of data center services to solve the automatic decision-making of traffic optimization. DRL-TE [59] leverages an efficient DRL-based control framework to solve the traffic engineering problem in communication networks. This paper extends the application of DRL to the app prediction.

Cellular Data. There are some studies using the same cellular network request data as our study [62], [63], [64], [65], [66], [67], [68]. SAMPLES [62] provides a framework to identify the application identity according to the network request by inspecting the HTTP header. CellSim [63] extracts similar trajectories from a large-scale cellular dataset. Yu *et al.* [64] present a city-scale analysis of app usage data on smartphones. TU *et al.* [65] re-identify a user in the crowd by the apps she uses and quantify the uniqueness of app usage. Wang *et al.* [66] discover users' identifiers in multiple cyberspace. However, the above studies do not leverage the app usage data for real-time app prediction.

8 Discussion

Limitations. DeepAPP has several limitations. First, the cellular data cannot capture the app usages that do not make any network requests or make requests through Wi-Fi networks. However, such a limitation does not impact the performance much. Since app usages collect from a large number of users, DeepAPP can still learn the general app usage behaviors of different users by the general agent. Moreover, DeepAPP updates the personalized agent based on the online app usages, which can cover all the apps the user opens.

Second, our system relies on back-end component for model training and updating, and such an over-the-top service hinders our system from being used on large scale users. However, with the help of current device-side deep learning developing tool TensorFlow Lite, we can still deploy our general prediction model on the smartphone for inference at the expense of accuracy degradation over time. In addition, it is encouraging that our system can be used as a standalone application in the future, where the hardware performance will be further enhanced and more powerful device-side deep learning developing frameworks can be developed.

Deployment. The deployment of DeepAPP is mainly associated with the expense of back-end infrastructure placement. The back-end component consists of two modules, i.e., the context database and two agents. As the kernel of DeepAPP, agents provide fast prediction model for user, which requires adequate computing resources (e.g., CPU) for the running of DeepAPP. Besides, context database provides the reservation of training samples and hence a reliable and effective storage system is available.

Discomfort Caused by Energy Consumption. DeepAPP will bring about power consumption of the mobile devices, especially when making inference on the front-end (as depicted in Fig. 21). This may affect the user's experience, e.g., overheat, decreased usage time, etc. To avoid that, we could leverage the DNN compression techniques (e.g., weight compression, convolution decomposition, etc) to reduce the model complexity. We leave it for future work.

Privacy Issues. In the data-driven evaluation, the data provider has anonymized the app usage data by replacing the user identification by a hash code. The app usage data only contain anonymized records of cell tower sequences, without any information relating to text messages, phone conversations or search contents. Besides, we randomly select from a large dataset for our dataset, which can also prevent leaking the mobile users' privacy.

In the field experiments, DeepAPP collects some private sensitive data (e.g., contextual information) from volunteers. To protect the privacy, we anonymize the user identifier in the database. In addition, since our context feature only need the POI distribution around the user, we do not need the exact location of the user.

9 Conclusion

This paper presents DeepAPP, a deep reinforcement learning framework for mobile app prediction, which predicts the next apps in the next time slot on her mobile device. By combining a context-aware state representation method, a personalized agent and a general agent together, DeepAPP can provide effective and efficient app prediction. Extensive data-driven evaluations and field experiments demonstrate high performance gain of DeepAPP.

Acknowledgments

Zhihao Shen and Kang Yang are contributed equally to this work.

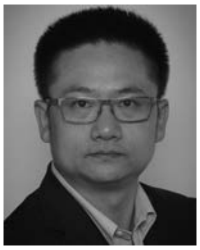
References

[1] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proc. Int. Conf. Mobile Syst. Appl. Serv.* 2012, pp. 113–126.
[2] X. Ye *et al.*, "Preference, context and communities:A multi-faceted approach to predicting smartphone app usage patterns," in *Proc. Int. Symp. Wearable Comput.*, 2013, pp. 69–76.
[3] D. T. Nguyen *et al.*, "Reducing smartphone application delay through read/write isolation," in *Proc. 13th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2015, pp. 287–300.
[4] A. Parate, M. Bhmer,, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2013, pp. 275–284.
[5] P. Baumann and S. Santini, "Every byte counts: Selective prefetching for mobile applications," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 2, pp. 1–29, 2017.
[6] Y. Wang, X. Liu, D. Chu, and Y. Liu, "Earlybird: Mobile prefetching of social network feeds via content preference mining and usage pattern analysis," in *Proc. 16th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 67–76.
[7] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff, "Context-aware application scheduling in mobile systems: What will users do and not do next?," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 1235–1246.
[8] H. Cao and M. Lin, "Mining smartphone data for app usage prediction and recommendations: A survey," *Pervasive Mobile Comput.*, vol. 37, pp. 1–22, 2017.
[9] N. Natarajan, D. Shin, and I. S. Dhillon, "Which app will you use next? Collaborative filtering with interactional context," in *Proc. 7th ACM Conf. Recommender Syst.*, 2013, pp. 201–208.
[10] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian, "Exploiting enriched contextual information for mobile app classification," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 1617–1621.
[11] K. Huang, C. Zhang, X. Ma, and G. Chen, "Predicting mobile application usage using contextual information," in *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 1059–1065.
[12] C. Shin, J. H. Hong, and A. K. Dey, "Understanding and prediction of mobile application usage for smart phones," in *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 1059–1065.
[13] Z. X. Liao, S. C. Li, W. C. Peng, P. S. Yu, and T. C. Liu, "On the feature discovery for app usage prediction in smartphones," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 1127–1132.
[14] R. Baezayates, D. Jiang, F. Silvestri, and B. Harrison, "Predicting the next app that you are going to use," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2015, pp. 285–294.
[15] X. Chen, Y. Wang, J. He, S. Pan, Y. Li, and P. Zhang, "CAP: Context-aware app usage prediction with heterogeneous graph embedding," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 3, no. 1, pp. 1–25, 2019.
[16] S. Xu *et al.*, "Predicting smartphone app usage with recurrent neural networks," in *Proc. Int. Conf. Wirel. Algorithms Syst. Appl.*, 2018, pp. 532–544.
[17] V. Kostakos, D. Ferreira, J. Goncalves, and S. Hosio, "Modeling smartphone usage: A Markov state transition model," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous*, 2016, pp. 486–497.
[18] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[19] H. Van Hasselt , A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[20] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 389–400.
[21] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," 2015, *arXiv:1512.07679*.
[22] B. SARWAR, "Item-based collaborative filtering recommendation algorithms," in *Proc. Int. Conf. World Wide Web*, 2001, pp. 285–295.
[23] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online learning: A comprehensive survey," 2018, *arXiv:1802.02871*.

[24] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
[25] TensorFlow Lite. [Online]. Available: https://tensorflow.google.cn/lite/
[26] WJX. [Online]. Available: www.wjx.cn
[27] AMAP [Online]. Available: https://www.amap.com
[28] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.
[29] M. Bhmer, B. Hecht, J. Schning, A. Krger, and G. Bauer, "Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage," in *Proc. Int. Conf. Hum. Comput. Interact. Mobile Devices Serv.*, 2011, pp. 47–56.
[30] T.-M.-T. Do and D. Gatica-Perez , "By their apps you shall understand them: Mining large-scale patterns of mobile phone usage," in *Proc. Int. Conf. Mobile Ubiquitous Multimedia*, 2010, pp. 1–10.
[31] C. Sun, J. Zheng, H. Yao, Y. Wang, and D. F. Hsu, "AppRush: Using dynamic shortcuts to facilitate application launching on mobile devices," *Procedia Comput. Sci.*, vol. 19, no. 19, pp. 445–452, 2013.
[32] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Ranking fraud detection for mobile apps: A holistic view," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 619–628.
[33] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Int. Conf. Mach. Learn.*, 2014, pp. I-387–I-395.
[34] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[35] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.
[36] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," 2015, *arXiv:1512.07679*.
[37] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Mach. Learn.*, vol. 8, no. 3–4, pp. 225–227, 1992.
[38] Code. [Online]. Available: https://github.com/kelleann/appPrediction
[39] F. Nie, H. Huang, X. Cai, and C. H. Ding, "Efficient and robust feature selection via joint l2, 1-norms minimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 1813–1821.
[40] H. Zhang *et al.*, "OnRL: Improving mobile video telephony via online reinforcement learning," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.
[41] S. Isaacman *et al.*, "Identifying important places in people's lives from cellular network data," in *Proc. Int. Conf. Pervasive Comput.*, 2011, pp. 133–151.
[42] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. 32, no. 4, pp. 396–402, Apr. 1984.
[43] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 1999, pp. 16–22.
[44] G. Albaum, "The likert scale revisited," *Market Res.Soc. J.*, vol. 39, no. 2, pp. 1–21, 1997.
[45] Accubattery. [Online]. Available: https://www.accubatteryapp.com/
[46] J. V. Joseph, J. Kwak, and G. Iosifidis, "Dynamic computation offloading in mobile-edge-cloud computing systems," in *Proc. IEEE Wirel. Commun. Netw. Conf.*, 2019, pp. 1–6.
[47] A. Ahmad, I. Alseadoon, A. Alkhalil, and K. Sultan, "A framework for the evolution of legacy software towards context-aware and portable mobile computing applications," in *Proc. Int. Conf. Softw. Eng. Res. Pract.*, 2019, pp. 3–9.
[48] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application energy metering framework for android smartphones using kernel activity monitoring," in *Proc. USENIX Conf. Annu. Techn. Conf.*, 2012, p. 36.
[49] Z. X. Liao, Y. C. Pan, W. C. Peng, and P. R. Lei, "On mining mobile apps usage behavior for predicting apps usage in smartphones," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 609–618.
[50] S. Zhao *et al.*, "AppUsage2Vec: Modeling smartphone app usage for prediction," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1322–1333.
[51] 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 *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 40–52.
[52] G. Huang *et al.*, "ShuffleDog: Characterizing and adapting user-perceived latency of android apps," *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2913–2926, Oct. 2017.
[53] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt , M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1995–2003.
[54] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, 1928–1937.
[55] G. Zheng *et al.*, "DRN: A deep reinforcement learning framework for news recommendation," in *Proc. World Wide Web Conf.*, 2018, pp. 167–176.
[56] T. Li, Z. Xu, J. Tang, and Y. Wang, "Model-free control for distributed stream data processing using deep reinforcement learning," *Proc.VLDB Endowment*, vol. 11, no. 6, pp. 705–718, 2018.
[57] C. Li, L. Justinas, C. Kai, and L. Feng, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. Conf. CM Special Interest Group Data Commun.* 2018, pp. 191–205.
[58] X. Ding, W. Du, and A. Cerpa, "OCTOPUS: Deep reinforcement learning for holistic smart building control," in *Proc. ACM Int. Conf. Syst. Energy-Efficient Buildings, Cities, Transp.*, 2019, pp. 326–335.
[59] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. Conf. Comput. Commun.* 2018, pp. 1871–1879.
[60] Z. Shao, W. Wu, Z. Wang, W. Du, and C. Li, "Seaships: A large-scale precisely annotated dataset for ship detection," *IEEE Trans. Multimedia*, vol. 20, no. 10, pp. 2593–2604, Oct. 2018.
[61] X. Ding, W. Du, and A. E. Cerpa, "MB2C: Model-based deep reinforcement learning for multi-zone building control," in *Proc. ACM Int. Conf. Syst. Energy-Efficient Buildings, Cities, Transp.*, 2020, pp. 50–59.
[62] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao, "SAMPLES:Self adaptive mining of persistent lexical snippets for classifying mobile application traffic," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 439–451.
[63] Z. Shen, W. Du, X. Zhao, and J. Zou, "Retrieving similar trajectories from cellular data at city scale," 2019, *arXiv:1907.12371*.
[64] D. Yu, Y. Li, F. Xu, P. Zhang, and V. Kostakos, "Smartphone app usage prediction using points of interest," *ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 1–21, 2018.
[65] Z. Tu *et al.*, "Your apps give you away: Distinguishing mobile users by their app usage fingerprints," *ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 2, no. 3, pp. 1–23, 2018.
[66] H. Wang, C. Gao, Y. Li, Z. Zhang, and D. Jin, "From fingerprint to footprint: Revealing physical world privacy leakage by cyberspace cookie logs," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 1209–1218.
[67] D. Li and Z. Shao, "The new era for geo-information," *Sci. China Series F: Inf. Sci.*, vol. 52, no. 7, pp. 1233–1242, 2009.
[68] Y. Wei, X. Zhao, J. Zou, and E. Herrera-Viedma, "A complementing preference based method for location recommendation with cellular data," *Knowl.-Based Syst.*, vol. 183, 2019, Art. no. 104889.

Zhihao Shen received the BE degree in automation engineering from the School of Electronic and Information, Xi'an Jiaotong University, Xi'an, China, in 2016, where he is currently working toward the PhD degree with the Systems Engineering Institute. His research interests include mobile computing and behavior computing.

Kang Yang received the BE degree in automation engineering from the School of Electrical and Control Engineering, Xi'an University of Science and Technology, Xi'an, China, in 2016, and the ME degree in control engineering from the School of Electronic and Information, Xi'an Jiaotong University, Xi'an, China, in 2019. He is currently working toward the PhD degree with the University of California, Merced. His research interests include mobile systems and wireless sensor network.



Xi Zhao (Senior Member, IEEE) received the PhD degree in computer science from the Ecole Centrale de Lyon, Ecully, France, in 2010. He conducted research in the fields of biometrics and pattern recognition as a research assistant professor with the Department of Computer Science, University of Houston, USA. He is currently a professor with Xi'an Jiaotong University, China. He is currently a Tang Scholar. His current research interests include mobile computing and behavior computing.

Jianhua Zou (Member, IEEE) received the bachelor's, master's, and doctor's degrees from the Huazhong University of Science in 1984, 1987, and 1991, respectively. He is currently a professor with Xi'an Jiaotong University. His researchinterests mainly include control systems and computer networks and multimedia.

Wan Du (Member, IEEE) received the BE and MS degrees in electrical engineering from Beihang University, China, in 2005 and 2008, respectively, and the PhD degree in electronics from the University of Lyon (Ecole Centrale de Lyon), France, in 2011. He is currently an assistant professor with the University of California, Merced. From 2012 to 2017, he was a research fellow with Nanyang Technological University, Singapore. His research interests include the Internet of Things, distributed networking systems, and mobile computing.