RLMob: Deep Reinforcement Learning for Successive Mobility Prediction

Ziyan Luo
Mila, McGill University
luo.ziyan@mila.quebec

Congcong Miao*
Tsinghua University
mccmiao@163.com

ABSTRACT

Human mobility prediction is an important task in the field of spatiotemporal sequential data mining and urban computing. Despite the extensive work on mining human mobility behavior, little attention was paid to the problem of successive mobility prediction. The state-of-the-art methods of human mobility prediction are mainly based on supervised learning. To achieve higher predictability and adapt well to the successive mobility prediction, there are four key challenges: 1) disability to the circumstance that the optimizing target is discrete-continuous hybrid and non-differentiable. In our work, we assume that the user's demands are always multi-targeted and can be modeled as a discrete-continuous hybrid function; 2) difficulty to alter the recommendation strategy flexibly according to the changes in user needs in real scenarios; 3) error propagation and exposure bias issues when predicting multiple points in successive mobility prediction; 4) cannot interactively explore user's potential interest that does not appear in the history. While previous methods met these difficulties, reinforcement learning (RL) is an intuitive answer for this task to settle these issues. We innovatively introduce RL to the successive prediction task. In this paper, we formulate this problem as a Markov Decision Process. We further propose a framework - RLMob to solve our problem. A simulated environment is carefully designed. An actor-critic framework with an instance of Proximal Policy Optimization (PPO) is applied to adapt to our scene with a large state space. Experiments show that on the task, the performance of our approach is consistently superior to that of the compared approaches [1].

CCS CONCEPTS

• **Information systems** → *Location based services*;

KEYWORDS

Human Mobility Prediction, Successive Mobility Prediction, Deep Reinforcement Learning, Sequential Data Mining

[1]Our code is available at https://github.com/SunsetRay/RLMob

* Corresponding author: Congcong Miao (mccmiao@163.com).

ACM Reference Format:
Ziyan Luo and Congcong Miao*. 2022. RLMob: Deep Reinforcement Learning for Successive Mobility Prediction. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3488560.3498438

1 INTRODUCTION

The advance in GPS and Wi-Fi-equipped mobile devices and location-acquisition systems have generated massive spatiotemporal trajectory data representing human mobility. A large amount of trajectory data provides us with unprecedented information (e.g., location, timestamp, user ID) to understand human mobility patterns and stimulates a number of trajectory mining tasks for various applications [30]. For example, Deepmove [5] and VANext [7] model sequential transition regularities from historical trajectory data and predict where a user will go next for personalized resource recommendation and allocation. TULER [8] and DeepTUL [19] learn the essential intricate features from the whole trajectory and then links the trajectory to users for identifying the potential criminals or terrorists.

Despite the extensive work on mining human mobility behavior, especially next location prediction [5, 7, 15, 18], little attention was paid to the problem of *successive mobility prediction* (i.e., predicting locations of the next few steps) – which could benefit both governments and individuals. For example, by successively predicting the future locations the human tends to visit, governments could estimate the evolution of traffic and design intelligent transportation systems with smart scheduling strategies in advance to handle the crowd aggregation and reduce the traffic jams and potential stampede [29]. In mobile edge computing scenarios where people always take advantage of cloud services to improve the computation capabilities and energy efficiencies of mobile devices, network operators always need to make the successive mobility prediction to constantly migrating the corresponding resources to cater to the movement of its user. Therefore, users could benefit from lower service latency and enjoy a better quality of experience [25]. Moreover, it could help in forecasting the trace of terrorists/criminals for public safety [8].

With the proliferation of such applications, it is in great need to solve the *successive mobility prediction* problem – which is a challenging task. The previous works on predicting human mobility all focus on the next location prediction by learning sequential statistical models. The Markov based models [17, 23] build a transition matrix between all locations to predict the next move of a user, while *recurrent neural network* (RNN) [5, 8] based methods learn sequential transition patterns in high-dimension space by location embedding to improve the prediction accuracy. Despite

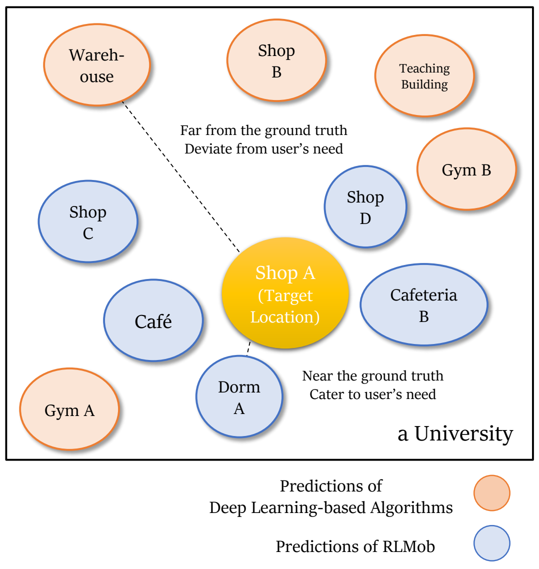

Figure 1: When the algorithm predicted a wrong location.

the promising results of RNN-based approaches for crowd flow prediction, there are four key challenges to be solved to realize high predictability of successive mobility prediction.

Firstly, in most cases, the locations that an algorithm predicts do not coincide with the true next locations. In such scenarios, the "wrong" locations are not always the same in the user's mind. However, in most supervised learning methods, *cross-entropy loss* is applied which equally regards all negative locations. Figure 1 depicts this phenomena. In Figure 1, the supervised learning approaches fail to consider this issue and the prediction deviates the user's need (only one in the correct category and all the predicted location are geographically far from the ground truth), while our approach achieves better user satisfaction (two points in the correct category and all the predicted location are geographically near the ground truth). In the above-mentioned case, the multi-targeted modeled user satisfaction can be more precisely modeled as a discrete-continuous hybrid function, which is non-differentiable and hard to be optimized with supervised learning methods. Analogously, the user satisfaction collected online is also multi-targeted which is more likely to be a discrete-continuous hybrid target. Supervised learning methods are hard to tackle such target functions.

Secondly, the user has a dynamically evolving preference which has long-term dependencies. For instance, a user used to buy staple goods at the shop A. However, for some reason, this shop recently stopped selling goods C that the user liked. For a long time, this user chose to go to the shop B that is far away from his home instead of the shop A. After some time, the shop A resumed to sell goods C and the user again, chose to purchase goods at the shop A. When dealing with such cases, supervised learning methods are not flexible. These approaches cannot capture the rapidly changing preference when interacting with users. Models should be constantly retrained using up-to-date data to keep on with the user's evolving preference.

Thirdly, especially in successive mobility prediction, the *exposure bias issue* cannot be ignored. This issue is often mentioned in natural language processing tasks, which means in test stage, the model has no exposure to the full range of errors when the teacher-forcing technique [27] is applied. If this technique is not used, the error of prediction could propagate which further hampers the algorithm to perform well.

Lastly, both traditional methods and supervised learning methods are lacking the exploration of user preference. Pretrained models can only predict mobility in a fixed manner.

The above-mentioned challenges can all be solved via *reinforcement learning* (RL), which is widely applied in many decision-making problems [2, 3, 16]. We innovatively introduce RL to successive mobility prediction. The first challenge can be settled because the reward in RL can be non-differentiable. In the case described in the second challenge, the interactive nature and the trial-and-error learning method of RL make it a promising solution in such a scenario. The third challenge is also not an issue in RL because the training sequences are generated by the models rather than fetching from ground truth using the teacher-forcing technique in RNN-based methods. As for the last challenge, exploration is always a significant part of RL algorithms. In our work, the agent learns a stochastic policy with some possibility to sample the non-optimal action. *Deep reinforcement learning* (DRL) combines RL with deep learning which has better generalizability.

In this work, our contribution can be summarized in four folds:

- We attack a novel problem *successive mobility prediction* (i.e., predicting locations of the next few steps) and leverage *deep reinforcement learning* (DRL) to conduct *successive mobility prediction*. There is few existing work that utilizes DRL to conduct successive mobility prediction. We formulate the successive mobility prediction problem as a *Markov Decision Process* (MDP).
- We further propose a framework - RLMob to solve the successive mobility prediction problem. A simulated environment (RLMob Environment Simulator) is carefully designed, which simulates user satisfaction by making full use of information in the dataset. It enables the agent to do offline training through interacting with the simulated environment. The state and reward are further carefully designed in order to better capture useful information and simulate user satisfaction.
- We apply *actor-critic* framework with an instance of *Proximal Policy Optimization* (PPO) to adapt to our scene with a large state space. *Generalized Advantage Estimation* (GAE) is further utilized to better shaping the advantage function. The pretraining is applied to the policy network (actor) to reduce the impact on random seeds and help converge.
- We conduct comparison experiments on traditional methods, state-of-the-art methods, and a basic DRL algorithm on a self-made dataset and two publicly available datasets. Results show that our method consistently superior to the state-of-the-art supervised learning methods and is also better than the basic RL-based method. It shows the superiority of DRL in our *successive mobility prediction* task and the effectiveness of our amendment.

2 RELATED WORK

Previous works dealing with human mobility prediction mainly focus on two categories of methods: pattern-based methods and model-based methods.

Pattern-based methods. These methods firstly find out some certain mobility patterns in history, and then predict mobility with these patterns. Most early works are based on *matrix factorization* (MF). Liu et al. [14] adopt *probabilistic non-negative matrix factorization*, learning geographical predilection based on user's mobility. Lian et al. [13] propose GeoMF model. It is based on *weight-based matrix factorization* (WMF) and the geographic impact.

Model-based methods. They aim to learn statistical models from the trajectories. Early methods are mostly based on *Markov chain* theory. These approaches establish transition matrices to help predict locations. *Recurrent neural network* (RNN) is a state-of-the-art model to capture sequential patterns. It has an outstanding ability to fit into the trajectory and session data mining tasks [10, 15]. Hidasi et al. develop GRU4Rec [10] which applies GRU to the session-based recommendation. Liu et al. [15] propose Spatial-Temporal Recurrent Neural Networks (ST-RNN) to model temporal and spatial contexts. The about-mentioned approaches work well on the trajectory data with considerable sequential patterns. However, the GTSM data is very sparse with high order sequential patterns, bringing big challenges to realize accurate mobility prediction. Miao et al. [18] apply the *convolutional neural network* (CNN) to capture sequential patterns as local features of the image that can tackle high order sequential patterns.

Reinforcement Learning (RL) is widely applied in many decision-making problems [2, 3, 16]. *Actor-critic* [24] is a commonly used architecture in RL that uses an actor to estimate the optimal policy and a critic to evaluate the performance of the actor. REINFORCE [26] is a basic form of policy optimization which increases the possibility of actions that have high rewards while decreasing the possibility of actions with low rewards. *Trust Region Policy Optimization* (TRPO) [20] is proposed to improve the stability and convergence of REINFORCE. It takes the largest step possible to improve performance while satisfying a *KL-Divergence* constraint on the distance between new and old policies. *Proximal Policy Optimization* (PPO) [22] simplifies the problem to first-order and utilizes some other tricks to keep the old and new policy close. PPO-Penalty and PPO-Clip are the two main variants of PPO.

3 PRELIMINARIES

3.1 Notations and Problem Formulation

The trajectory data contains millions of entries in a region. Each entry is always recorded with spatial, temporal, and personal information to represent the essential elements for an event. We partition a region (e.g. city or campus) in a suitable coordinate system (e.g., categories, longitude, latitude or some landmarks) to get representative location points, namely *point of interest* (POI). As for temporal information, motivated by previous work [18, 19], we align all the real-value timestamps in the entry into a fixed time interval to aggregate mobility statistics and then split the time interval into T timeslots to learn the context information. Let $\mathcal{P}$ denotes a set of POIs and $\mathcal{U}$ denotes a set of users.

Definition 1. *(Trajectory Sequence [18]) We define a spatiotemporal point q as a tuple of location point p and timeslot t, i.e., $q = (p, t)$. For a user ID u, trajectory sequence $\mathcal{T}$ is the aggregation of spatiotemporal points, i.e., $\mathcal{T}_u = q_1 q_2 \cdots q_L$, where q is listed by the timestamp.*

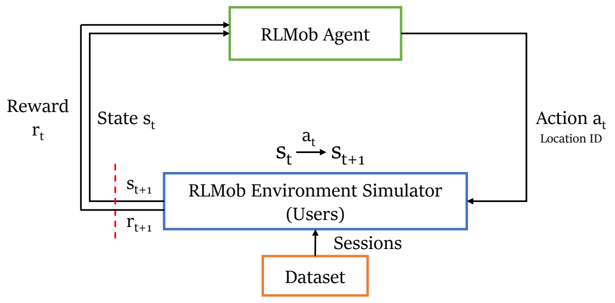

Figure 2: The interactions between RLMob Agent and RL-Mob Environment in MDP.

Definition 2. *(Trajectory) Given a trajectory sequence $\mathcal{T}_u$ for a user u, trajectory is a subsequence of $\mathcal{T}_u$. The k-th trajectory with length m can be represented as $\mathcal{T}_u^k = q_k q_{k+1} \cdots q_{k+m-1}$.*

Transforming a trajectory sequence into several trajectories does not only reduce computational complexity but also enables us to mine richer knowledge. Since it is not the core part of our work, following [18], we use the sliding window method with length m to generate the trajectories. As for successively predicting next n locations with corresponding timestamp set, we transfer the set of timestamp into the timeslot set, represented as $T=\{t_1, t_2, \cdots, t_n\}$.

Problem (Successive Mobility Prediction): Given historical trajectory $\mathcal{T}_u^k = q_k q_{k+1} \cdots q_{k+m-1}$, and the prediction timeslot set T_u^k, the task of successive mobility prediction is to predict the next n spatial points, i.e., $p_{k+m} p_{k+m+1} \cdots p_{k+m+n-1}$, which is called the *target session.*

Here n and m can be variables. Thus, the user historic trajectory and target session can both be variable-length.

3.2 MDP Formulation

The problem of *successive mobility prediction can be modeled* as a *Markov Decision Process* (MDP). MDP is defined by the five-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ the transition function, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function. At every time step, the RLMob agent interacts with the environment, taking an action $a \in \mathcal{A}$ according to the current state $s \in \mathcal{S}$ and receives a reward R. This trial-and-error search procedure is illustrated in Figure 2. We did not reduce the problem into a *partially observed MDP* (POMDP) to avoid over-complication. The details of each element are as follows:

- **State $\mathcal{S}$**. At each time step, an agent requires an observation of the environment to determine what to do next. In this work, the following observations may helpful: a user's characteristics, user historical trajectory, and the timeslots. Details of state definition are in Section 4.1.
- **Action $\mathcal{A}$**. An action a is to recommend a location in the action space. The action space of the RLMob agent is all the available locations in the dataset, which is a discrete action space. To simplify the problem, at each time step, the RLMob agent only recommends one location to the user.
- **Transition function $\mathcal{P}$**. It defines the probability of transition from the current state s to the next state s'. Once the RLMob agent takes an action and the next timeslot is collected, the

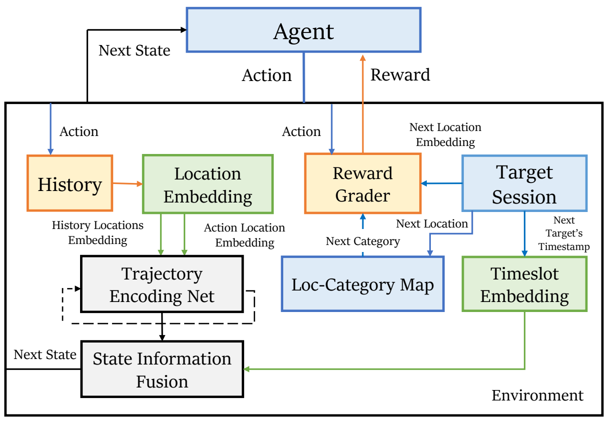

Figure 3: The architecture of RLMob Environment Simulator and its interaction with the agent.

transition $p(s'|s, a)$ is determined. More details can be seen in the next section.

- **Reward** $\mathcal{R}$. When the RLMob agent takes an action $a \in \mathcal{A}$ at a state $s \in \mathcal{S}$, the RLMob environment simulator will give an immediate feedback $R(s, a)$ to evaluate the action a at state s. The details of the reward design can be seen in the next section.
- **Discount factor** γ. The discount factor $\gamma \in [0, 1]$ helps the RLMob agent strike a balance between the instant feedback and the long-term yield. $\gamma = 0$ implies RLMob only considers instant reward and $\gamma = 1$ implies all future rewards are fully counted.

4 METHOD

4.1 RLMob Environment Simulator

In this section, we mainly introduce a simulated environment for our framework to do offline training. To initialize the environment properly, pretraining for locations, users, timeslots embeddings and trajectory encoding net is essential. The well-designed state and reward enable the environment to cooperate with the agent to complete the *successive mobility prediction* task.

4.1.1 Environment Initialization. First, we need to convert the real-value timestamp of each record (i.e. the number of seconds since 1970) into the fixed-interval timeslot. According to the concept of timeslots in Section 3.1, we divided all the timestamps into 168 timeslots, each of which has a length of 1 hour, representing each hour of the week. We delete some users with too short trajectories to ensure there is enough data for the algorithm to exploit. Subsequently, we split the dataset into training and test dataset. Next, we divide the user trajectory sequences into a fixed-length or unfixed-length trajectory using the sliding window algorithm. It not only improves the utilization of data but also adapts the session length to the capacity of the models.

Another essential step is pretraining. All the embeddings are pretrained using GRU4Rec [10] under the task of predicting the next location.

With all these steps taken, all preprocessed datasets except for the target part of the test dataset are feed into the RLMob environment simulator.

4.1.2 State Components. The components of the state are designed as follows:

The characteristics of the users e_u. In particular, when it comes to datasets with a large range of positions, different users have significant differences in their preferences, which are closely related to users' life habits, demographic information, and other factors. But in some cases, such as a university mobility dataset, the user behavior pattern is monotonous, and this feature may become noise.

The historical trajectory $\mathcal{T}_u^k = q_k q_{k+1} \cdots q_{k+m-1}$ of the user. It is the most important part of the state, which contains user preferences and some high-level mobility patterns. It is the most useful information to predict the following locations and should be part of the state representation in any case. *The timeslots of historical locations* may also make sense, because it adds contextual information and, in some cases, allows an agent to better understand the spatiotemporal relationship. However, this will also greatly increase the complexity of the state space. Considering the weak convergence of DRL algorithms, such characteristics may not be applicable in all cases. In our framework, the mentioned user trajectories are encoded by the *trajectory encoding net*, a pretrained *long short-term memory* (LSTM), which outputs a fixed-length trajectory representation vector.

The prediction timeslot set T_u^k. It may also be helpful because users generally have different behavior patterns at different times, such as going to the restaurant for dinner at 6:30 PM every night, going to the library to read books at 9:00 AM on Sunday, etc. In RLMob, one of the following timeslot is used in each state.

Pretrained embeddings. We adopt the representation of the features after pretraining as the state representation inspired by Item2Vec [1]. We pretrain all the embedding using GRU4Rec [10] under the task of predicting the next location.

4.1.3 Reward Design. The reward function in RL is a feedback value given by the environment to an agent to evaluate the agent's actions. The optimization goal of RL is to maximize the expected cumulative return. The goal of this work is to maximize user satisfaction with the successive predicted locations, and it is intuitive to model user satisfaction within the reward function. But how to define “satisfaction” and, to what extent the user “satisfies”? It usually requires complex modeling to make it accurately depicted. A simple approach is to directly apply the framework online and ask users how satisfied they are with the predicted results, or infer their satisfaction from some of their behaviors. However, this approach requires a large amount of cost and loss of users caused by the cold-start issue of the recommender system.

Therefore, in this work, we did not adopt this approach but designed a function to approximate the user's demand. In this work, we believe that better predictions are not only to predict more “optimal” locations but also to make the deviation from the “optimal” locations smaller when the agent makes a mistake. An illustration of the idea is shown in Figure 1. Thus, the function describing user needs is designed into a discrete-continuous hybrid function. Note that this reward function can also be substituted for other user satisfaction metrics that are calculated more reasonably, while other parts of the framework can still be applied.

In this paper, the reward function $r(s_t, a_t)$ is designed as follows:

$$r(s_t, a_t) = PS(s_t, a_t) + CS(s_t, a_t) + L2\text{-}Dist(s_t, a_t) \tag{1}$$

where *PS* indicates the score of whether the prediction aligns with the ground truth:

$$PS(s_t, a_t) = \begin{cases} k, & a_t = l^*(s_t) \\ 0, & a_t \neq l^*(s_t) \end{cases} \tag{2}$$

where $l^*(s_t)$ is the ground truth obtained from the target session and k is a constant.

CS indicates the score of whether the prediction is align with the true location's category:

$$CS(s_t, a_t) = \begin{cases} b, & C(a_t) = c^*(s_t) \\ 0, & C(a_t) \neq c^*(s_t) \end{cases} \tag{3}$$

where $c^*(s_t)$ is true location's category, $C(a_t)$ is the category of predicted location (action) a_t and b is a constant.

L2-Dist is the scaled *Euclidean Distance* between pretrained embeddings of the predicted location and the ground truth, which represents the similarity of locations in "context". s_t and a_t respectively represents the state and action at the time step t.

$$L2\text{-}Dist(s_t, a_t) = \alpha \frac{\sqrt{\sum_{i=1}^{|D|}(e_L^i(a_t) - e_{l^*}^i(s_t))^2}}{|D|} \tag{4}$$

where α is the factor to balance the scale of *Dist* and the other two scoring function *PS* and *CS*, $|D|$ is the dimension of the location embedding vector, $e_L^i(a_t)$ is the i-th feature in the prediction location's embedding vector, $e_{l^*}^i(s_t)$ represents the i-th feature in the true location's embedding vector from the target session.

Another issue in the reward design is that, the variance of the reward function is large. It tends to lead to the gradient instability, making the models difficult to converge. We can utilize RNN-based method to predict an action a_t' as our baseline with reward $r(s_t, a_t')$. The improved reward $R(s_t, a_t)$ is given by:

$$R(s_t, a_t) = r(s_t, a_t) - r(s_t, a_t') \tag{5}$$

4.1.4 Architecture of RLMob Environment Simulator. The architecture is shown in Figure 3. At first, the environment gets an action from the agent (if it is at the initial step of an episode, no action is needed). The action becomes part of the *history*, which is encoded by the *trajectory encoding net*. With the next timeslot embedding fused, the next state is determined. The action is also fed into the *reward grader*, which grades the action using the reward in Section 4.1.3. The reward is then returned to the agent.

4.2 RLMob Agent

It is challenging to predict successive locations with DRL. The heterogeneity of user trajectory with unfixed length, diversified of preference and even some serendipity leads to a large variety of state encoding, which makes our agent difficult to converge.

Thus, in this work, we adopt the *actor-critic* architecture [24] with an instance of *Proximal Policy Optimization* (PPO) [22] to solve the problems above. Utilizing *actor-critic* framework brings us flexibility to choose a reasonable horizon to update the value function and the policy to accelerate convergence.

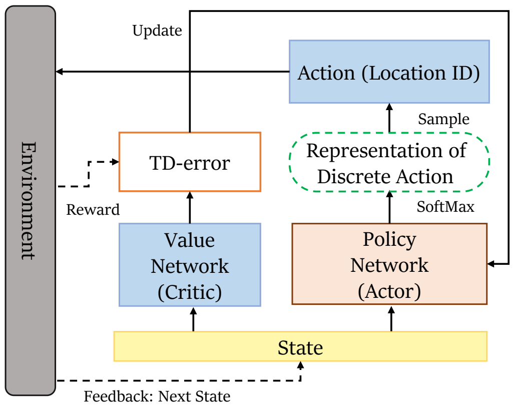

Figure 4: The architecture of RLMob Agent. We leverage the actor-critic framework.

The agent's actor-critic architecture and its interaction with the environment are shown in Figure 4. The framework has a policy network to estimate the policy π_θ, and a value network called critic to compute the state-value function $V_\phi(s_t)$. In the framework, the actor network's parameters θ are updated as follows:

To accelerate training, the actor network can also be pretrained. The pretraining also reduce the impact of different model initialization methods and random seeds. To help pretraining, the structure of the network can be the same as the fusion part of our baseline, GRU4Rec, introduced in Section 5.2. Thus, our method can also be regarded as an enhancement of the pretrained network.

PPO is one of the state-of-the-art policy optimization approach and is applied in a wide range of problems. PPO-Penalty and PPO-Clip are the two main variants of PPO. In this work, we mainly refer to PPO-Clip. It minimizes the following objective to learn a stochastic policy π_θ:

$$\begin{aligned} J_{PPO-Clip}(\theta) = \sum_{(s_t,a_t)} \min \Bigg(& \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \\ & \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s_t, a_t)\Bigg) \end{aligned} \tag{6}$$

where ϵ is the hyperparameter that controls the distance between old policy π_{θ_k} and new policy π_θ. The advantage function $A^\pi(s_t, a_t)$ is estimated using the *Generalized Advantage Estimation* (GAE) [21] as follows:

$$A(s_t, a_t) = \sum_{l=0}^{L} (\gamma\lambda)^l \delta_{t+l}^V \tag{7}$$

where L represents the future horizon used to estimate the advantage, γ is the discount factor and λ is a hyperparameter representing state trade-off between bias and variance. δ_t^V is the TD-error:

$$\delta_t^V = R(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{8}$$

where V_ϕ is the aforementioned state-value function, which is updated by:

$$J(\phi) = \sum_{l=0}^{L} (\delta_t^V)^2 \tag{9}$$

4.3 The Training Procedure

The whole training procedure of the framework is depicted in Figure 4 and Algorithm 1. In Algorithm 1 and 2, lines with ◇ are executed by the RLMob agent, and lines with ▲ are executed by the RLMob environment simulator. In Algorithm 1, the horizon L represents the future steps to estimate the advantage and indicates how many steps the models update. The gradient step K indicates how many times the gradient is updated using one batch of data.

Algorithm 1 RLMob Training Stage

```
input: ◇ Initial parameters $\theta, \phi$
  for each episode do
    ▲ Initialize the environment with a random trajectory
    ▲ Make the initial state $s_0$

    for horizon $L$ do
      for each environment step do
        ◇ Sample action from the policy    $a_t \sim \pi_\theta(a_t|s_t)$
        ▲ Compute the reward $r_t$ (See Section 4.1.3)
        ▲ Combine $a_t$ to the user historic trajectory
        ▲ Make the next state $s_{t+1}$ (See Section 4.1.2)
        ◇ Store the transition    $(s_t, a_t, r_t, s_{t+1}, done)$
      end for

      for gradient step $K$ do
        ◇ Take out the transition(s)    $(s_t, a_t, r_t, s_{t+1}, done)$
        ◇ Update the critic parameters    $\phi \leftarrow \phi - \lambda_{V_\phi} \nabla_\phi J(\phi)$
        ◇ Update policy weights    $\theta \leftarrow \theta - \lambda_{\pi_\theta} \nabla_\theta J(\theta)$
      end for
    end for
  end for
output: ◇ Optimized parameters $\theta, \phi$
```

4.4 The Testing Procedure

After some episodes of training, the performance of RLMob is examined and then the framework continues to alternate between the training state and the test stage. The test stage is similar to the training stage. The main difference is that in the test stage we select the action with the max probability directly rather than sampling. This is because, in the offline testing procedure, there is no need for exploration which can impair the overall performance. The procedure of the offline test of our method is listed in Algorithm 2.

5 EXPERIMENT

In this section, we study the performance of the RLMob agent on the successive mobility prediction problem. We build a dataset based on the Wi-Fi data collected at a university and use two publicly available datasets to test our agent. We perform various comparison experiments to study the effectiveness of the purposed method.

Algorithm 2 RLMob Test Stage

```
for each episode (item in test dataset) do
  ▲ Initialize the environment with the historic trajectory
  ▲ Make the initial state $s_0$

  for each environment step do
    ◇ Get action from the policy    $a_t = \arg\max_{a_t} \pi_\theta(a_t|s_t)$
    ▲ Compute the reward $r_t$ (See Section 4.1.3)
    ▲ Combine $a_t$ to the user historic trajectory
    ▲ Make the next state $s_{t+1}$ (See Section 4.1.2)
    ▲◇ Make statistics on $a_t$
  end for
end for
```

Table 1: The statistics of preprocessed datasets. $|\mathcal{U}|$: number of users; $|\mathcal{P}|$: number of locations; $|\mathcal{C}|$: number of categories; $|\mathcal{R}|$: average length of trajectory sequences; $|\mathcal{S}|$: number of checkins (data items).

Datasets	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{S}\rvert$
F-TKY	1729	300	32	98.96	171094
F-NYK	532	500	120	79.91	42514
Univ-WIFI	10966	114	7	17.44	75021

5.1 Experimental Settings

Dataset. We conduct experiments on three datasets. Foursquare[2] datasets are publicly available geotagged social media (GTSM) datasets with high sparsity, provided by [28]. Experiments are conducted on the subset of Foursquare datasets of two cities, Tokyo (F-TKY) and New York (F-NYK). Additionally, we collect the Wi-Fi data of mobile phones at a university and build a dataset called "Univ-WIFI" (information is properly anonymized and collected on a voluntary basis), and experiment on the complete dataset.

Details about Data Preprocessing. We apply the method described in Section 4.1.1 to preprocess the dataset. To make all our comparison approaches applicable, every processed user trajectory contains fixed length of historical trajectory m and target sessions n ($m = 7$ and $n = 5$). For all datasets, we filter some invalid data items and directly split them into training and test datasets with the split rates 0.8 and 0.2 respectively. Only frequently visited locations are retained to make the action space in a decent scale and ensure sufficient training of their embeddings. Datasets are preprocessed to make the action spaces in different scales for testing in different settings. Details of preprocessed datasets can be seen in Table 1.

Evaluation Metrics. Since the human mobility prediction problem can be reduced to a multi-classification problem, we adopted common metrics for multi-classification. However, these metrics cannot evaluate negative results. Thus the evaluation combines our self-designed metrics with those common metrics as follows:

[2]https://sites.google.com/site/yangdingqi/home

- **Episode Final Return**. *Episode Final Return* means the average final return (sum of all rewards in an episode) over the test dataset, which is an intuitive performance indicator of RL algorithms and a comprehensive metric of the simulated user satisfaction. It is also the direct optimization goal of RLMob, thus it is the most important metric among our metrics. The definition of the reward function is in Section 4.1.
- **Acc@1**. *Acc@1* is the ratio of a method's correct prediction, which directly indicates performance. "Correct" means the ground truth location $l^*(s)$ is the same as the prediction result $L(s)$. The formulation can be represented as:

$$Acc@1 = \frac{|\{s \in S : l^*(s) = L(s)\}|}{|S|} \tag{10}$$

  where $|S|$ is the test dataset size and s is a data item.
- **Macro-F1**. *Macro-F1* is the harmonic mean of *Macro-P* and *Macro-R* which is averaged across all locations:

$$\text{Macro-F1} = \frac{2 \times \text{Macro-P} \times \text{Macro-R}}{\text{Macro-P} + \text{Macro-R}} \tag{11}$$

  where Macro-P is the macro-precision and Macro-R is the macro-recall.
- **CoCiN**. *CoCiN* stands for "Correction of Category in Negative results". It represents that even if the agent's prediction is wrong, how much the predicted location's category $C(s)$ can at least match the category of the ground truth $c^*(s)$. It can be formulated as follows:

$$CoCiN = \frac{|\{s \in S : c^*(s) = C(s), l^*(s) \neq L(s)\}|}{|S|} \tag{12}$$

  where $|S|$ is the test dataset size and s is a data item.
- **L2-Dist**. *L2-Dist* is the scaled *Euclidean Distance* between pre-trained embeddings of the predicted location and the ground truth. It portrays the deviation of ground truth and the wrong prediction. Details see Section 4.1.

Implementation Details. To reduce randomness, all our experiments are repeated three times with different random seeds. To simulate the online scene, our RLMob agent is evaluated on the test dataset after 5,000 episodes of training on the training dataset.

We empirically find that different user in Univ-WIFI dataset has similar mobility patterns while the other two Foursquare datasets have highly personalized user trajectories. Arbitrarily adding user information to help prediction for Univ-WIFI causes worse performance. Thus, user information is not used only in all experiments of Univ-WIFI dataset.

Hyperparameters Tuning. In our experiments, some key hyperparameters are as follows. k in Equation 2 is set to 20, b in Equation 3 is set to 4 and α in Equation 4 is set to 0.2. For all DNN-based methods, the learning rate is set to 1e-3, which is set to 1e-5 for all RL-based methods. The discount factor γ is set to 0.5 and λ in Equation 7 is set to 0.8. The clip coefficient ϵ in Equation 6 is set to 0.1. In Algorithm 1, the horizon L is set to 5 and the gradient step K is set to 3. *Adam* optimizer [12] is applied in all supervised learning and RL-based approaches.

5.2 Comparison Approaches

The settings of all the approaches are as follows:

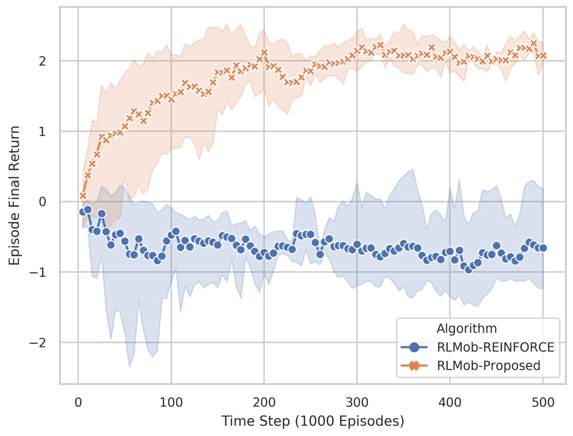

Figure 5: The results of Episode Final Return of RL-based methods on Univ-WIFI. The shade area represents the range of the values in duplicate experiments. The agents are evaluated after 5,000 episodes of training, and a solid dot in the figure represents the evaluation result.

- **MC**. *Markov chain* (MC) [6] regards all the visited locations as states and builds a transition matrix between these states to make predictions on human mobility.
- **MF**. *Matrix factorization* (MF) [14] is commonly applied in recommender systems. It takes human mobility prediction as a recommendation procedure, factorizing the users-locations matrix to generate user's preferences and make predictions.
- **DNN-based methods**. We implement *multi-layer perceptron* (MLP), namely *deep neural network* (DNN) with backpropagation as our baseline. The inputs of the MLP are embeddings of historic locations and output are the probabilities of consecutive target locations. The user information is used as the auxiliary information in prediction, and the timeslot set is not added due to information leakage.
- **RNN-based methods**. We refer to GRU4Rec [10] as for RNN-based methods. It applies RNN to the session-based recommendation. To adapt it to our scene, we apply the *teacher-forcing* technique [27] to extend the prediction to multiple points. We empirically find that *long short-term memory* (LSTM, [9, 11]) performs better than *gated recurrent unit* (GRU, [4]) in our settings and apply LSTM instead of GRU, which is different in recommender system. In addition to LSTM, a fusion layer that fuses the user information and one target timeslot in timeslot sets is jointed. This method is utilized as our *baseline* in Section 4.1.3 to reduce the variance of reward. Note that RLMob can be regarded as an enhancement to the deep learning (DL) methods. Other state-of-the-art DL methods can also be enhanced by RLMob in a similar way by replacing the network structure, thus we only compare GRU4Rec.
- **RLMob-REINFORCE**. REINFORCE [26] is a *Monte-Carlo policy gradient* method, which is a basic form of policy-based RL algorithms. The fundamental idea is that it increases the possibility of actions that have high rewards while decreasing the possibility of actions with low rewards.

Table 2: The performance of all comparison approaches. Improvement indicates the improvement of our method compared with GRU4Rec because GRU4Rec is the strongest baseline in general. This table only shows best results of all methods.

Dataset	Metric/Method	MC	MF	MLP	GRU4Rec	RLMob-REINFORCE	**RLMob-Proposed**	Improvement
Univ-WIFI	**Episode Final Return**	-8.315	-20.93	-0.6380	0.0000	0.2329	2.191	-
	Acc@1	0.1350	0.0118	0.2079	0.2110	0.2127	0.2291	8.58%
	Macro-F1	0.0472	0.0093	0.2015	0.1565	0.1564	0.1664	5.95%
	CoCiN	0.1093	0.2745	0.1489	0.1568	0.1590	0.1745	10.14%
	L2-Dist	1.142	1.861	1.223	1.189	1.186	1.185	0.34%
F-TKY	**Episode Final Return**	-6.730	-22.73	-0.5780	0.0000	0.2229	2.397	-
	Acc@1	0.2705	0.1842	0.3381	0.3931	0.3948	0.4150	5.57%
	Macro-F1	0.2551	0.1535	0.3266	0.3532	0.3449	0.3602	1.98%
	CoCiN	0.3023	0.0000	0.2664	0.0031	0.0036	0.0036	16.13%
	L2-Dist	0.7933	1.059	0.7714	0.7026	0.6939	0.6623	5.74%
F-NYK	**Episode Final Return**	-3.907	-39.59	-8.027	0.0000	0.0800	0.4402	-
	Acc@1	0.3977	0.0755	0.3599	0.4362	0.4369	0.4401	0.73%
	Macro-F1	0.4266	0.0781	0.3853	0.4377	0.4407	0.4469	0.89%
	CoCiN	0.0170	0.0299	0.0211	0.0036	0.0037	0.0040	7.5%
	L2-Dist	1.250	1.944	1.334	1.185	1.185	1.177	0.68%

5.3 Performance Evaluation

The performance of all comparison approaches is shown in Table 2. Note that: it is meaningless to compute improvement on *Episode Final Return* because it is a relative value. *Episode Final Return* on GRU4Rec is always zero because we take it as a baseline in the reward function. *Improvement* indicates the improvement of our method compared with GRU4Rec because GRU4Rec is the strongest baseline in general. Besides, curves on *Episode Final Return* of two RL-based approaches on Univ-WIFI dataset are provided in Figure 5 as an example. Results on the other two datasets are similar.

By and large, MF performs the worst on almost all evaluation metrics. Trivially reduce our successive mobility prediction problem to a recommendation problem is unpromising, as it cannot capture the user's mobility patterns. Besides, it can only recommend one location without considering any spatial or sequential information. MC performs better than MF because it considers the sequential order of user trajectories. However, the non-Markov nature of user trajectories makes MC a weak baseline.

As for DNN-based methods, MLP generally performs the worst. Many previous works empirically show that RNN is a better estimator than MLP when handling sequential data and GRU4Rec as expected, performs better than MLP.

In general, RL-based methods perform the best on user satisfaction (*Episode Final Return*) and other metrics as we expected. RL-based methods circumvent the exposure bias issue of GRU4Rec, thus they outperform other approaches on the metric *Acc@1*. RL-based methods also can directly optimize the reward function which represents user satisfaction. As a result, the metrics of *L2-Dist* and *CoCiN* of RL-based are both improved compared to GRU4Rec. Results in Figure 5 and Table 2 prove that RLMob-Proposed performs better than RLMob-REINFORCE on both effectiveness, stability, and convergence. As many previous works proved that PPO has better convergence than REINFORCE, it fits better with our scene. We also find that our agent performs the worst on F-NKY. This may be owing to the relatively large action space and data sparsity (weak correlation of two consecutive points) of this dataset.

6 DISCUSSION

Currently, we only run experiments on our RLMob environment simulator, which cannot strictly reflect user satisfaction in real production. To further optimize it, a real online test is needed. With real user feedbacks, the reward function representing user satisfaction can be modeled more accurately. Another important future work of us is to solve this problem in the dataset with a larger action space. It is non-trivial because of the weak convergence of DRL algorithms and the overestimation issue for problems with exceedingly large action space. Moreover, to apply our framework in real production, the inference time of the models should be better studied. To make the system's feedback instant, some optimization like model compression should be carefully considered.

7 CONCLUSION

In this work, we attack the *successive mobility prediction* problem, and innovatively leverage *deep reinforcement learning* (DRL) to solve the problem. We reduce the problem into a *Markov Decision Process* (MDP). We also design the RLMob framework and describe the interaction flow between the framework and the simulated environment. Some advanced DRL algorithms that can be applied to this framework like PPO are introduced. Furthermore, through experiments, we prove that the method proposed in this work solves the difficulties of optimization objective non-differentiability, the dynamic nature of user demand, and exposure bias when predicting successive locations. The performance of our method is consistently better than other compared approaches.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No. 2020YFE0200500.

REFERENCES

[1] Oren Barkan and Noam Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 1–6.

[2] Yanju Chen, Chenglong Wang, Osbert Bastani, Isil Dillig, and Yu Feng. 2020. Program Synthesis Using Deduction-Guided Reinforcement Learning. In *Computer Aided Verification*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 587–610.

[3] Wei Cheng, Ziyan Luo, and Qiyue Yin. 2021. Adaptive Prior-Dependent Correction Enhanced Reinforcement Learning for Natural Language Generation. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 14 (May 2021), 12701–12709. https://ojs.aaai.org/index.php/AAAI/article/view/17504

[4] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[5] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. DeepMove: Predicting Human Mobility with Attentional Recurrent Networks. (2018), 1459–1468.

[6] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. 2012. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*. ACM, 3.

[7] Qiang Gao, Fan Zhou, Goce Trajcevski, Kunpeng Zhang, Ting Zhong, and Fengli Zhang. 2019. Predicting Human Mobility via Variational Attention. (2019), 2750–2756.

[8] Qiang Gao, Fan Zhou, Kunpeng Zhang, Goce Trajcevski, Xucheng Luo, and Fengli Zhang. 2017. Identifying human mobility via trajectory embeddings. (2017), 1689–1695.

[9] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. 2015. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455* (2015).

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. 2014. GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 831–840.

[14] Bin Liu, Yanjie Fu, Zijun Yao, and Hui Xiong. 2013. Learning geographical preferences for point-of-interest recommendation. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1043–1051.

[15] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts. In *Thirtieth Aaai Conference on Artificial Intelligence*.

[16] Ziyan Luo, Linfeng Zhao, Wei Cheng, Sihao Chen, Qi Chen, Hui Xue, Haidong Wang, Chuanjie Liu, Mao Yang, and Lintao Zhang. 2021. Match Plan Generation in Web Search with Parameterized Action Reinforcement Learning. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, New York, NY, USA, 1040–1052. https://doi.org/10.1145/3442381.3449862

[17] Wesley Mathew, Ruben Raposo, and Bruno Martins. 2012. Predicting future locations with hidden Markov models. In *Proceedings of the 2012 ACM conference on ubiquitous computing*. ACM, 911–918.

[18] Congcong Miao, Ziyan Luo, Fengzhu Zeng, and Jilong Wang. 2020. Predicting Human Mobility via Attentive Convolutional Network. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 438–446.

[19] Congcong Miao, Jilong Wang, Heng Yu, Weicheng Zhang, and Yinyao Qi. 2020. Trajectory-User Linking with Attentive Recurrent Network. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*. 878–886.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[21] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR* abs/1506.02438 (2016).

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[23] Hongzhi Shi, Hancheng Cao, Xiangxin Zhou, Yong Li, Chao Zhang, Vassilis Kostakos, Funing Sun, and Fanchao Meng. 2019. Semantics-Aware Hidden Markov Model for Human Mobility. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 774–782.

[24] RS Sutton and AG Barto. 1998. Reinforcement Learning: An Introduction Vol. 1 MIT press.

[25] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, and Kin K Leung. 2019. Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision Process. *IEEE/ACM Transactions on Networking* PP, 99 (2019).

[26] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[27] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[28] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. 2015. Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 1 (2015), 129–142.

[29] Junbo Zhang, Yu Zheng, and Dekang Qi. 2016. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. (2016), 1655–1661.

[30] Yu Zheng. 2015. Trajectory Data Mining: An Overview. *Acm Transactions on Intelligent Systems Technology* 6, 3 (2015), 1–41.