Next Point-of-Interest Recommendation on Resource-Constrained Mobile Devices

Qinyong Wang
The University of Queensland
qinyong.wang@uq.edu.au

Hongzhi Yin*
The University of Queensland
h.yin1@uq.edu.au

Tong Chen, Zi Huang
The University of Queensland
{tong.chen@,huang@itee.}uq.edu.au

Hao Wang
Alibaba AI Labs
cashenry@126.com

Yanchang Zhao
Data61, CSIRO
yanchang.zhao@csiro.au

Nguyen Quoc Viet Hung
Griffith University
quocviethung1@gmail.com

ABSTRACT

In the modern tourism industry, next point-of-interest (POI) recommendation is an important mobile service as it effectively aids hesitating travelers to decide the next POI to visit. Currently, most next POI recommender systems are built upon a cloud-based paradigm, where the recommendation models are trained and deployed on the powerful cloud servers. When a recommendation request is made by a user via mobile devices, the current contextual information will be uploaded to the cloud servers to help the well-trained models generate personalized recommendation results. However, in reality, this paradigm heavily relies on high-quality network connectivity, and is subject to high energy footprint in the operation and increasing privacy concerns among the public. To bypass these defects, we propose a novel Light Location Recommender System (LLRec) to perform next POI recommendation locally on resource-constrained mobile devices. To make LLRec fully compatible with the limited computing resources and memory space, we leverage FastGRNN, a lightweight but effective gated Recurrent Neural Network (RNN) as its main building block, and significantly compress the model size by adopting the tensor-train composition in the embedding layer. As a compact model, LLRec maintains its robustness via an innovative teacher-student training framework, where a powerful teacher model is trained on the cloud to learn essential knowledge from available contextual data, and the simplified student model LLRec is trained under the guidance of the teacher model. The final LLRec is downloaded and deployed on users' mobile devices to generate accurate recommendations solely utilizing users' local data. As a result, LLRec significantly reduces the dependency on cloud servers, thus allowing for next POI recommendation in a stable, cost-effective and secure way. Extensive experiments on two large-scale recommendation datasets further demonstrate the superiority of our proposed solution.

ACM Reference Format:
Qinyong Wang, Hongzhi Yin*, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. 2020. Next Point-of-Interest Recommendation on Resource-Constrained Mobile Devices. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380170

*Corresponding author and having equal contribution with the first author.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380170

1 INTRODUCTION

Recently, next point-of-interest (POI) recommender systems are emerging, which aim to recommend a list of POIs which are the most likely to be visited by the target user based on his/her current spatiotemporal contexts. They can not only fulfill users' spatiotemporal preferences with personalized recommendations, but also effectively help nearby merchants attract more customers. On the other hand, as many mobile social platforms like Instagram and Foursquare have gained immense popularity, many users are encouraged to share their locations and experiences, leading to a massive amount of accumulated geo-tagged data. With the availability of such geo-tagged data, plentiful approaches based on Deep Neural Networks (DNNs) have been proposed to model users' long- and short-term preferences and mobility patterns, providing more accurate next POI recommendations. For example, ST-LSTM [48] incorporates spatiotemporal contexts into the Long- and Short-term Memory (LSTM) networks to model users' sequential behaviors, while TCMA [16] introduces the multi-level temporal context attention mechanisms to dynamically select the relevant check-ins and discriminative contextual factors to predict users' next destinations. Compared with traditional models based on Matrix Factorization (MF) [17] or Markov Chain [27, 50], these DNN-based methods achieve the state-of-the-art performance owing to strong representation capability and model expressiveness.

1.1 Next POI Recommendation Services

Though most existing next POI recommender systems are based on DNNs, their prominent effectiveness comes with the notoriously high cost of abundant computing resource and large-scale training data. Consequently, current DNN-based next POI recommender systems have to work in a purely **cloud-based** paradigm to train and deploy the next POI recommender models for two major reasons. One is the *powerful computing processors* on the cloud. Training deep models is a computationally expensive process that requires plenty of multi-core CPUs/GPUs to substantially reduce the training time. The other reason is the need for *large data storage*. Access to large-scale datasets is critical for DNN-based models to perform accurate and timely recommendation. Under certain privacy agreements, massive users' historical data, sensor data, POI information and other contextual information are collected from the mobile devices, and are only stored and processed in the servers.

Once the training completes, the cloud-based recommender systems serve users in a pipeline manner. When a user requests for advice on the next travel, his/her mobile device first transfers user identification, current time and location, as well as other sensor data to the cloud servers. Afterwards, based on the data received and relevant information retrieved from the cloud storage, the could-based recommender systems generate a personalized POI list, which will be sent back to the user's mobile device and then displayed.

Nevertheless, this cloud-based paradigm encounters three major issues when applied in the real-world scenarios. The first problem is its high dependency on the network quality. Due to the nature of cloud services, stable and fast network connectivity is crucial for transmitting requests and recommendation results between servers and mobile devices. This fundamental requirement, however, cannot always be met in many situations. For example, during public holidays, overcrowded tourist attractions are subject to limit the Wi-Fi bandwidth speed for each connected mobile device, making users to suffer from high latency when using cloud-based POI recommendation services. In worse cases, some tourist attractions are located in remote areas with extremely limited telecom infrastructures, where the cloud-based services become futile as users' mobile devices go off-line. Secondly, running cloud-based next POI recommender systems is environmentally unfriendly in terms of carbon emissions. To maintain the service quality and information processing speed, the energy cost and carbon footprint for the bandwidth and computing resources from cloud server providers are extremely high, especially during peak periods. As a result, the cloud-based paradigm poses high environmental pressure on recommendation service platforms. The third problem comes from the privacy concerns. Not all the users are willing to share their personal data with recommendation platforms due to privacy issues. Moreover, data privacy has been legalized by many recent laws, e.g., General Data Protection Regulation (GDPR) enforced by the Europe Union in 2018. On this occasion, a lot of crucial but sensitive data like users' check-in history and personal profile can no longer be uploaded to the cloud servers for analysis and storage, which will eventually impede the performance of personalized recommendation.

To tackle the above issues of purely cloud-based next POI recommender systems, we propose a **local paradigm**, which is a novel recommendation scheme allowing DNN-based recommendation models to operate locally on users' mobile devices. In our local paradigm, a next POI recommendation model is firstly trained on the resource-rich cloud servers, which will be downloaded by mobile devices when the network is in good condition (e.g., Wi-Fi connected) and starts to operate locally. As such, the local scheme allows for instant next POI recommendation on mobile devices irrespective of the network connectivity. Since cloud servers are now only responsible for providing a trained recommendation model, the energy cost for the expensive cloud servers is also reduced. Additionally, all users' private data can be safely retained on their own devices because the recommendation model is deployed locally on users' mobile devices.

1.2 Challenges and Our Solution

Despite the promising benefits, mobile devices are extremely constrained in computing and storage resources compared with cloud servers, making the local recommendation paradigm a non-trivial goal. Next, we outline the primary challenges and our solution.

Efficiency on Mobile Devices. On resource-constrained mobile devices such as cellphones and smart watches, apart from lowering the time complexity of the model, the space complexity, i.e., the sheer amount of model parameters, should also be compatible with the limited memory space. However, while the latest next POI recommendation models based on DNNs, especially RNNs, dominate in terms of recommendation performance, their space complexity increases by an order of magnitude compared with traditional models. Moreover, a typical RNN-based model usually involves frequent data exchange and memory allocation to calculate the final result, which incurs more energy and memory consumption when a user's data accumulates over time. let us take the aforementioned TCMA model as an example. It has at least six embedding matrices, two LSTM networks and two fully connected layers. Consequently, its parameter size can easily reach 100 Megabytes (Mb) when processing a typical POI dataset with more than one million users and one million POIs. This makes existing next POI recommendation models hardly transferrable to mobile devices.

Recommendation Quality and Model Robustness. Though we can effectively reduce the computational complexity and parameter size by devising a compact model structure, we do not expect compromises on the recommendation performance. On one hand, previous studies [6, 29] show that light and shallow models have a higher risk of failing to deliver optimal predictions in contrast to deep networks. On the other hand, we have very limited control on data quality for a locally deployed model. For instance, the auxiliary information about POIs like other users' reviews and tags are inaccessible, and we have little knowledge about a user's long-term preferences due to restricted storage capacity. So, another challenge arises when we try to maintain the robustness of the locally deployed model so that high-quality recommendation results can still be generated with lightweight methods and limited data.

Our Solution. To tackle the above challenges, we first propose **L**ight **L**ocation **Rec**ommender System (LLRec), a compact next POI recommendation model to cope with the first challenge. LLRec takes a recently proposed variant of RNN named FastGRNN [14] as its main building block. As FastGRNN extends the residual connection to a gate by reusing the low-dimensional RNN matrices, it is able to match the effectiveness of state-of-the-art gated RNNs like Gated Recurrent Unit (GRU) and LSTM but with significantly lower time and space complexity. Moreover, we identify that the bottleneck of reducing model size lies in the enormous embedding layers, which are the dominating part of the total parameters. Therefore, we further compress the model by introducing the tensor-train format [23], which has been successful in many compression tasks such as video classification [39] and word embedding [12], to represent the transformation matrices in the embedding layer. Eventually, we can store the embedding matrices with much fewer parameters. To prevent our smaller and shallower model LLRec from underperforming, we innovatively introduce two strategies that enable LLRec to produce high-quality recommendation results with light model structure and highly constrained data availability. Despite the promising performance of RNNs in modelling behavioral sequences [47], the decision of the next POI to visit is typically determined by the spatiotemporal factors [18, 48]. So, we firstly grant

the FastGRNN in LLRec the capability of handling spatiotemporal information. Specifically, we introduce time-specific and distance-specific transition matrices to the vanilla FastGRNN to characterize the spatiotemporal correlations between two adjacent check-ins. At the same time, to deal with the limited data, we leverage the knowledge distillation framework [10] to train the deployed model LLRec. The core idea is to encourage the small-scale student model, i.e., LLRec, to be finally deployed on mobile devices, to not only learn from the data, but also mimic the behaviors of the powerful teacher model. Thus, we develop a deep and complex teacher model that runs on the cloud servers. The teacher model extends the idea of LLRec but is also more complex, where a recurrent component is designed to model short-term spatiotemporal dependencies between adjacent activities, which is coupled with an attention component that learns user preferences based on historical check-ins with the attention mechanism. As the teacher model can access the rich data source from the cloud, it can capture the complex dependencies among user preferences. Eventually, with the knowledge distilled by the well-trained teacher model and passed to the student model, the simplified student model LLRec is able to perform accurate recommendations on mobile devices by merely using the recent check-in sequences without excessive side information.

Our major contributions are summarized as follows.

- We identify the major drawbacks of the widely used cloud-based next POI recommendation services in the tourism industry. To overcome these drawbacks, we propose a novel solution that can be locally deployed on mobile devices to perform timely and accurate next POI recommendation. To the best of our knowledge, we are the first to address how to run a recommender system, in particular a next POI recommender system, on mobile devices in the real-world scenario.
- To adapt to the resource-constrained nature of mobile devices, we propose LLRec, which is a lightweight and spatiotemporal next POI recommender system based on FastGRNN. In LLRec, we adopt the tensor-train decomposition to further compress the parameters in the embedding layer. Besides, we employ the knowledge distillation training framework to improve the prediction quality with the assistance of a more powerful and deeper teacher model.
- We validate the performance of LLRec in multiple aspects with two real-world datasets, and the experimental results fully demonstrate the superiority of our proposed model.

2 PRELIMINARIES

Before describing our proposals, we first introduce some important notations used in this paper and formalized our major tasks.

Let $\mathcal{U}$ and $\mathcal{V}$ be the set of users and POIs, respectively. Each POI $v \in \mathcal{V}$ is associated with the geographical coordinates (lon_v, lat_v).

Definition 1 (Plain Check-in Activity Sequence) The basic input unit of the next POI recommendation problem is the *check-in activity*, denoted by a triplet $x_t = (u, v, t)$, where $u \in \mathcal{U}$, $v \in \mathcal{V}$, and t is the timestamp for the check-in. A plain check-in activity sequence X contains T consecutive check-in activities by a user within a period of time, i.e., $X = [x_1, x_2, \ldots, x_T]$.

There are usually contextual information accompanied with a check-in activity, which can be utilized to mitigate the data sparsity problem commonly observed in the POI check-in history data. Examples of contextual information include user demographics, properties of POIs (e.g., categories and textual descriptions) and check-in contexts (e.g., weather conditions). So, we augment the check-in activity triplet with additional contextual information.

Definition 2 (Contextual Check-in Activity Sequence) A *contextual check-in activity* tuple is defined as $\bar{x}_t = (u, v, t, a)$, where a is the side information associated with this check-in. Accordingly, we define a contextual check-in activity sequence $\bar{X} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_T]$, and all contextual check-in activities by the same user u are represented by $\bar{X}^u = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{|\bar{X}^u|}]$, where $|\bar{X}^u|$ is length of the check-in sequence. For the same user u, $\bar{X}$ is a subset of $\bar{X}^u$ ($T \leq |\bar{X}^u|$), and likewise, $\bar{X}^u$ can also be divided into multiple $\bar{X}$ based on time periods.

Task 1 (Next POI Recommendation) Given X or $\bar{X}$, the *next POI recommendation* task aims to predict k POIs that are most likely to be visited by this user at the next time step.

Naturally, a next POI recommendation model trained with contextual check-in activity sequences could produce more accurate POI recommendation than its counterpart trained with plain check-in activity sequences, but the model complexity and computing resource consumption will also increase. On this basis, we introduce the second task.

Task 2 (Model Training) Given a powerful model *teacher* and a compressed lightweight model *student* on the cloud servers, we first train *teacher* utilizing contextual check-in activity sequences. Then, we train *student* with both the plain check-in activity sequences and the knowledge transferred from the well-trained *teacher* while maintaining its small size.

3 THE LLREC MODEL

In what follows, we detail the designs of LLRec. Owing to the limitation of storage and computing resources on mobile devices, we develop LLRec following three principles. (I) To save processing time and memory space, LLRec only has access to the most recent check-ins of the device's owner instead of his/her complete history. In real-world scenarios, it is impractical to assume its instant access to the side information stored on the cloud, hence it should be able to learn the latent sequential patterns with only the plain check-in activity sequence X as input. (II) It should prioritize the spatiotemporal factor to adapt to the spatiotemporal dynamics of user interests for timely POI recommendation. (III) The model should be compactly sized and computationally efficient to reside in mobile devices, which requires a balanced trade-off among the model size, efficiency and prediction accuracy.

3.1 Model Description

In the next POI recommendation task, users' next visit is largely dependent on their most recent (i.e., short-term) preferences [4, 48], which are reflected by the latest visited POIs. For example, it is highly possible for users to visit a restaurant nearby right after shopping in a supermarket. Based on sequential data, the state-of-the-art techniques [20, 46, 48] for modeling short-term user preferences are based on RNN, especially its variants LSTM and

GRU with the gating mechanism. Inspired by existing explorations, LLRec is also built upon a recurrent architecture to capture users' dynamic short-term preferences. Different from these methods, as we aim to combine simplicity and effectiveness for LLRec on resource-constrained mobile devices, we utilize the recently proposed cost-efficient recurrent network FastGRNN [14] as the building block. In particular, FastGRNN has nearly the same parameter size and inference complexity as the standard RNN, while also benefits from the gating mechanism [14].

3.1.1 Basic Model Structure. First, we transform the input data for LLRec. Given a plain check-in activity sequence X that contains T consecutive check-in activities by a user within a certain time interval: $X = [x_1, x_2, \ldots, x_T]$, where $x_t = (u, v, t)$, only the POI ID in x_t is considered as input feature due to the storage limit. For v in x_t, its D-dimensional latent vector representation $\mathbf{v}$ is retrieved via a look-up operation on the embedding matrix $\mathbf{W^v}$. Simply put, the input at time step t is $\mathbf{x_t}$ that equals to $\mathbf{v}$.

At time step t, the H-dimensional hidden state vector $\mathbf{h_t}$ in FastGRNN is updated by a weighted residual connection between the last hidden state $\mathbf{h}_{t-1}$ and the candidate hidden state $\hat{\mathbf{h}}_\mathbf{i}$:

$$\mathbf{h_t} = (\zeta(1 - \mathbf{z_t}) + \nu) \odot \hat{\mathbf{h}}_\mathbf{t} + \mathbf{z_t} \odot \mathbf{h_{t-1}} \tag{1}$$

where $0 \leq \zeta$ and $\nu \leq 1$ are trainable scalar parameters. The H-dimensional $\mathbf{z_t}$ and $\hat{\mathbf{h}}_\mathbf{t}$ are calculated based on the current input and the last hidden state:

$$\mathbf{z_t} = \sigma(\mathbf{W_x x_t} + \mathbf{W_h h_{t-1}} + \mathbf{b_z}) \tag{2}$$

$$\hat{\mathbf{h}}_\mathbf{t} = \tanh(\mathbf{W_x x_t} + \mathbf{W_h h_{t-1}} + \mathbf{b_h}) \tag{3}$$

where $\sigma(\cdot)$ is the Sigmoid function and $\tanh(\cdot)$ is the hyperbolic tangent function. Note that FastGRNN reuses the transformation matrices $\mathbf{W_z}$ and $\mathbf{W_h}$ for the vector-valued gating function in order to minimize the number of parameters and increase the efficiency.

Finally, we make recommendation based on the hidden state at the last time step T. Specifically, we adopt a bi-linear similarity comparison method introduced in [15] that takes up less memory space than using a fully-connected layer commonly seen in similar models. Based on a user's check-in sequence, a similarity score s_i is computed for each candidate POI v_i:

$$s_i = \mathbf{v_i B h_T} \tag{4}$$

where $\mathbf{v_i}$ is the D-dimensional vector representation of v_i and $\mathbf{B}$ is a trainable $D \times H$ matrix. The top-k POIs with the highest similarity scores then form a recommendation list.

3.1.2 Spatiotemporal Transition Gates. The plain check-in activity sequences can hardly offer comprehensive signals on users' dynamic preferences. The time intervals and the distances between successively visited POIs have substantial influence for the current and future prediction [16, 18, 21], which is called *spatiotemporal transition*. To this end, we further incorporate the spatiotemporal transition into LLRec. Specifically, we introduce the representations of the temporal interval and geographical distance between two successive check-in POIs v_t and v_{t-1} as Δt^t and Δg^t, respectively. Note that Δg^t can be calculated as the Euclidean distance:

$$\Delta g^t = \| \, lon_{v_t} - lon_{v_{t-1}}, lat_{v_t} - lat_{v_{t-1}} \, \|_2 \tag{5}$$

If we learn every possible continuous time interval and geographical distance representation, we would face the data sparsity problem. We follow the method proposed in [18] to tackle it. First, we quantize all timestamps and geographical distances into discrete slots, whose latent embeddings are represented in matrices $\mathbf{T}$ and $\mathbf{G}$. For a given Δt^t and Δg^t, their D-dimensional latent representations $\mathbf{\Delta t^t}$ and $\mathbf{\Delta g^t}$ are the interpolation of upper and lower bounds of their values:

$$\mathbf{\Delta t^t} = \frac{\mathbf{T}_{up(\Delta t^t)}[up(\Delta t^t) - \Delta t^t] + \mathbf{T}_{lo(\Delta t^t)}[\Delta t^t - lo(\Delta t^t)]}{up(\Delta t^t) - lo(\Delta t^t)} \tag{6}$$

$$\mathbf{\Delta g^t} = \frac{\mathbf{G}_{up(\Delta g^t)}[up(\Delta g^t) - \Delta g^t] + \mathbf{G}_{lo(\Delta g^t)}[\Delta g^t - lo(\Delta g^t)]}{up(\Delta g^t) - lo(\Delta g^t)} \tag{7}$$

where $up(\cdot)$ and $lo(\cdot)$ denote the upper and lower slots of the given values, respectively. In the end, we encode the spatiotemporal influence as implicit information into the gates of FastGRNN to guide the learning process [13, 18]. Eq 2 and Eq 3 are extended to be:

$$\mathbf{z_t} = \sigma(\mathbf{W_z x_t} + \mathbf{W_h h_{t-1}} + F_z(\mathbf{\Delta t^t}, \mathbf{\Delta g^t}) + \mathbf{b_z}) \tag{8}$$

$$\hat{\mathbf{h}}_\mathbf{t} = \tanh(\mathbf{W_z x_t} + \mathbf{W_h h_{t-1}} + F_h(\mathbf{\Delta t^t}, \mathbf{\Delta g^t}) + \mathbf{b_h}) \tag{9}$$

where $F_z(.)$ and $F_h(.)$ are add operations:

$$F_z(\mathbf{\Delta t^t}, \mathbf{\Delta g^t}) = \mathbf{W_{tz}} \mathbf{\Delta t^t} + \mathbf{W_{gz}} \mathbf{\Delta g^t} \tag{10}$$

and

$$F_h(\mathbf{\Delta t^t}, \mathbf{\Delta g^t}) = \mathbf{W_{th}} \mathbf{\Delta t^t} + \mathbf{W_{gh}} \mathbf{\Delta g^t} \tag{11}$$

where $\mathbf{W_{tz}}, \mathbf{W_{gz}}, \mathbf{W_{th}}, \mathbf{W_{gh}} \in \mathbb{R}^{H \times D}$ are time- and distance-specific transition matrices for the respective gates.

3.2 Model Compression

Unfortunately, neural models, including our LLRec, tend to suffer from an excess of parameters, i.e., there are massive redundancies in the model parameters that would consume huge storage and computing resources, thus we further compress the model by removing redundancies without compromising the performance.

In neural models, it is obvious that the weight matrices contribute much more to the parameter size compared with the vectorized biases, so we only focus on compressing the weight matrices in LLRec. To have a closer look at the case of our LLRec, we show the sizes of all trainable matrices with an example in Table 1, from which we can see that the POI embedding matrix $\mathbf{W^v}$ is dominating the parameter size. Actually, it takes up about 95% of the total parameters. To this end, our primary target is to reduce the size of the POI embedding matrix. The most intuitive solution is simple low-rank matrix factorization methods such as Singular Value Decomposition (SVD). Formally, those methods decompose a given matrix $\mathbf{E} \in \mathbb{R}^{I \times J}$ to a product of two matrices $\mathbf{UV^\intercal}$ with lower rank R: $\mathbf{U} \in \mathbb{R}^{I \times R}$ and $\mathbf{V} \in \mathbb{R}^{J \times R}$. As a result, the free parameters are reduced from IJ to $(I + J)R$, achieving a compression rate $\frac{IJ}{(I+J)R}$. However, embedding matrices usually have much more rows (representing the number of entries) than columns (representing embedding dimensions), i.e., $J \ll I$, consequently, directly applying these methods will prevent us from achieving a significant compression ratio because their compression ratio is bounded by $\frac{J}{R}$. Instead, we employ the recently proposed tensor-train decomposition [23, 24, 39], which allows for more compact representations. As a result, the number of parameters can be reduced to logarithmic w.r.t. the number of matrix rows.

Table 1: Example of trainable parameter sizes in LLRec. Assuming a typical scenario where the number of POIs $|\mathcal{V}|$ is 10,000, the embedding size D is 128, the number of hidden states H is 64 and the number of discrete time intervals and spatial distances is 50 and 150, respectively, the total number of parameter is therefore 1,350,456.

Matrix	$\mathbf{W^v}$	$\mathbf{W_x}$	$\mathbf{W_h}$	T	G	$\mathbf{W_{[t,g][z,h]}}$
Size	1,280,000	8,192	4,096	6,200	19,200	32,768

Before we present our model compression approach, we briefly introduce the tensor-train (TT) decomposition. Given a d-dimensional tensor $\mathcal{W} \in \mathbb{R}^{N_1 \times N_2 \times \ldots \times N_d}$, each entry indexed by $(l_1, l_2, \ldots, l_d)$ can be factorized into a sequence of matrix multiplications, which is termed TT-format:

$$\mathcal{W}(l_1, l_2, \ldots, l_d) = \mathcal{G}_1[l_1] \ldots \mathcal{G}_d[l_d] \tag{12}$$

where $\{\mathcal{G}_k\}_{k=1}^d$ are called TT-cores whose elements are matrices:

$$\mathcal{G}_k \in \mathbb{R}^{N_k \times R_{k-1} \times R_k}, R_k \in [1, N_k], R_0 = R_d = 1 \tag{13}$$

The sequence of ranks $\{R_k\}_{k=0}^d$ is called TT-rank.

Furthermore, assuming that the dimension size N_k can be factorized as $N_k = I_k \cdot J_k$, we can reshape $\mathcal{G}_k$ into $\mathcal{G}_k^* \in \mathbb{R}^{I_k \times J_k \times R_{k-1} \times R_k}$. The index l_k in Eq 12 thus can be represented with two indices (i_k, j_k) such that $l_k = i_k \cdot j_k$. Correspondingly, the factorization for tensor $\mathcal{W} \in \mathbb{R}^{(I_1 \cdot J_1) \times (I_2 \cdot J_2) \times \ldots \times (I_d \cdot J_d)}$ in Eq 12 can be rewritten as:

$$\mathcal{W}((i_1, j_1), \ldots, (i_d, j_d)) = \mathcal{G}_1^*[i_1, j_1] \ldots \mathcal{G}_d^*[i_d, j_d] \tag{14}$$

Next, we introduce the computation of the embedding for a particular POI v_i indexed by i in the space-consuming embedding matrix $\mathbf{W_v} \in \mathbb{R}^{|\mathcal{V}| \times D}$ based on the TT-format. First, we assume that $|\mathcal{V}| = \prod_{k=1}^d I_k$ and $D = \prod_{k=1}^d J_k$. Then, we map the row index i to d-dimensional vectors $\mathbf{i}$: $i \leftrightarrow \mathbf{i} = (i_1, \ldots, i_d)$, following the mapping procedures proposed in [12]. Finally, we compute all components of the embedding according to Eq 14. This process of computing all components is equivalent to selecting the particular slices in TT-cores, namely slices of shapes $J_1 \times R_1$ in $\mathcal{G}_1$, $R_1 \times J_2 \times R_2$ in $\mathcal{G}_2$ and so on, and performing a sequence of matrix multiplication [12].

Instead of explicitly storing the full tensor $\mathcal{W}$ of size $\prod_{k=1}^d I_k J_k = |\mathcal{V}| \cdot D$, now we only store its TT-format, i.e., the set of low-rank TT-cores $\{\mathcal{G}_k\}_{k=1}^d$ of size $\sum_{k=1}^d I_k J_k R_{k-1} R_k$, with which we can approximately reconstruct $\mathcal{W}$. Hence, the compression rate is:

$$rate = \frac{\prod_{k=1}^d I_k J_k}{\sum_{k=1}^d I_k J_k R_{k-1} R_k} \tag{15}$$

From this formulation, we can see that it is important to well tune the hyper-parameter TT-rank, i.e., if it is small, the model gets lighter but is also restricted from learning more complex representations, on the other hand, if it is large, the model has more flexibility but sacrifices its efficiency.

Using the same example in Table 1, we compare the number of matrix weights before and after tensor-train decomposition with TT-rank of 3, 4 and 5, and the corresponding compression rates (measured by Eq 15) in Table 2. We can see that the embedding parameter number decreases dramatically. If necessary, e.g., the devices are more demanding for the storage consumption, the model could be further compressed by tensorizing other matrices or choosing smaller TT-ranks.

Table 2: Example size of the embedding matrix before and after compression with different TT-ranks, and their compression rates, assuming that input dimension of 10,000 is factorized into $4 \times 25 \times 25 \times 4$ and the 128-dimensional embedding is factorized into $4 \times 4 \times 4 \times 2$.

Before	TT-Ranks	After	Compression Rate
1,280,000	3	1872	684
	4	3296	388
	5	5112	250

4 TRAINING LLREC

In this section, we describe how to train LLRec so that it can operate on resource-constrained mobile devices afterwards.

4.1 Learning Target

Compared with mobile devices, the cloud servers show strong advantages in computing capacity and richer data sources. As such, a recommender system trained and deployed on the cloud is more powerful and accurate. However, instead of following the conventional recommendation paradigm that generates recommendation results with well-trained and cloud-based systems, we propose to fully utilize the strengths of cloud servers to assist training our locally deployable model LLRec and optimize its performance.

Motivated by this idea, we design a teacher-student training scheme based on the knowledge distillation framework [10], which aims to transfer the knowledge from a complex and cumbersome teacher model to a small and simple student model. Intuitively, the rationale of this framework is to have the student model (i.e., our proposed LLRec) trained by not only learning from the plain check-in sequence data X but also understanding how the teacher model represents and works with the data. Specifically, assuming the student model and teacher model are respectively parameterized by θ and $\theta_{\mathcal{T}}$, the overall training loss of the student model for a single sequence is:

$$\mathcal{L}_{\mathcal{S}} = \lambda \mathcal{L}_{SL}(\theta) + (1 - \lambda) \mathcal{L}_{KD}(\theta, \theta_{\mathcal{T}}) \tag{16}$$

where $\mathcal{L}_{SL}$ is the loss for letting the student individually learn from the plain check-in sequence data X while $\mathcal{L}_{KD}$ is the loss for letting the student learn under the guidance of the teacher, and λ is a hyper-parameter to balance these two terms.

For $\mathcal{L}_{SL}$, we choose the popular Bayesian personalized ranking (BPR) pair-wise loss [26], which pairs a positive POI sample with a negative POI sample and compares the ranks (i.e., similarity scores) between the two POIs, then enforces that the rank of the positive POI should be higher than that of the negative POI. Formally, the loss $\mathcal{L}_{SL}$ is defined as:

$$\mathcal{L}_{SL}(\theta) = -\frac{1}{N_S} \sum_{j=1}^{N_S} \log \left(\sigma(s_i - s_j) \right) \tag{17}$$

where N_S is the number of negative samples, s_i and s_j are the scores for the positive POI v_i and the negative sampled POI v_j.

In the rest of this section, we will introduce $\mathcal{L}_{KD}(\theta, \theta_{\mathcal{T}})$ in detail.

4.2 Teacher Model

In this section, we first introduce the teacher model called $\mathcal{M}_t$ under the knowledge distillation framework. We propose two design principles for the teacher model. (I) It can make the most of the rich auxiliary information to address the data sparsity problem for the next POI recommendation. (II) With high tolerance to computational cost, its deep and multiplex neural architecture should effectively extract both long- and short-term user preferences from the abundant user history data on the cloud. According to the principles, we present the details of $\mathcal{M}_t$ as follows.

Owing to the easy access to rich data sources on the cloud, the input for $\mathcal{M}_t$ is extended to: (1) the recent contextual check-in activities $\bar{X}$; and (2) the full contextual check-in activity sequence $\bar{X}^u$ of this user. The input layer first transforms the POI v and side information a into their latent representations $\mathbf{v}$ and $\mathbf{a}$, respectively. Note that $\mathbf{a}$ may contain multiple data types and we need to apply corresponding feature extraction approaches for them. Take the commonly used textual contents as an example, for all words used to describe a POI, we convert them into low-dimensional embeddings via Word2Vec [22]. In the experiments, we use a pre-trained large-scale word embedding matrix from Google[1] to ensure the performance and efficiency. The mean of all description words' embeddings is calculated as the final representation of the textual feature for each POI. Then, we concatenate $\mathbf{v}$ and $\mathbf{a}$ as an aggregated input vector, which is then projected to a shared feature space via a feed-forward neural network, denoted as $\bar{\mathbf{x}}_\mathbf{t}$.

In $\mathcal{M}_t$, we employ a hierarchical structure with two neural components $\mathcal{M}_t^R$ and $\mathcal{M}_t^A$ to separately model $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}^u$, which respectively cover short- and long-term user preferences. Specifically, we want $\mathcal{M}_t^R$ to capture the short-term spatiotemporal dependencies between adjacent activities in $\bar{X}$, and $\mathcal{M}_t^A$ to capture the long-term user preferences in $\bar{X}^u$:

$$\mathbf{h}_\mathbf{T}^\mathbf{R} = \mathcal{M}_t^R(\bar{X}) \qquad \mathbf{h}_\mathbf{T}^\mathbf{A} = \mathcal{M}_t^A(\bar{X}^u) \tag{18}$$

where $\mathbf{h}_\mathbf{T}^\mathbf{R}$ and $\mathbf{h}_\mathbf{T}^\mathbf{A}$ are the hidden states of the last recurrent step from the corresponding components. They are then fused via weighted concatenation:

$$\mathbf{h}'_\mathbf{T} = w^R \mathbf{h}_\mathbf{T}^\mathbf{R} \oplus w^A \mathbf{h}_\mathbf{T}^\mathbf{A} \tag{19}$$

where w^R and w^A are hyper-parameters controlling the weights of two components. So far, $\mathbf{h}'_\mathbf{T}$ captures the user's long-term preference and short-term spatiotemporal transition patterns, which is used to make recommendations based on the similarity score s_i for each candidate POI v_i generated by a bi-linear function described in Eq 4 used in the student model LLRec. Finally, we also train the teacher model using a pair-wise loss function, similar to the student model.

The short-term modeling component $\mathcal{M}_t^R$ is trivial and inherits the design philosophy from the student model LLRec except three differences. First, it has larger representation capacities, i.e., we set a larger embedding size and have more hidden units for optimal performance. Second, it does not need to be compressed due to sufficient memory in cloud-based servers. Third, without constraints on computing and data resources, it adopts an additional computation process for contextual representation learning.

[1]https://code.google.com/archive/p/word2vec/

Next, we introduce the long-term modeling component $\mathcal{M}_t^A$. For check-in history data $\bar{X}^u$, the earlier a check-in is, the weaker influence it has on a user's next visit. Therefore, when modeling users' long-term preferences, the relative positions and the spatiotemporal transitions of their check-ins no long play such an important role in contrast to short-term preference modeling. To this end, we employ the attention mechanism [31, 36] to model long-term user preferences by dynamically selecting relevant POIs from the check-in history $\bar{X}^u$ of user u. Specifically, we utilize an item-level attention mechanism to compute a weighted sum of all hidden states learned from the check-in sequence $\bar{X}^u$:

$$\mathbf{h}_\mathbf{T}^\mathbf{A} = \sum\nolimits_{i=1}^{|\bar{X}^u|} \alpha_i \bar{\mathbf{x}}_\mathbf{i} \tag{20}$$

where the attention weight α is posed on different historical check-in activities to either emphasize or weaken their contributions to the final prediction. Each attention weight α_i is computed by quantifying the affinity between each historical check-in $\bar{\mathbf{x}}_\mathbf{i}$ and the final one $\bar{\mathbf{x}}_\mathbf{T}$ through dot product:

$$\alpha_i = \sigma\left(\frac{\bar{\mathbf{x}}_\mathbf{T} \cdot \bar{\mathbf{x}}_\mathbf{i}}{\sqrt{D'}}\right) \tag{21}$$

where D' is the dimension of $\bar{\mathbf{x}}_\mathbf{i}$ and $\bar{\mathbf{x}}_\mathbf{T}$.

It is worth mentioning that $\mathcal{M}_t^A$ can be removed when user history data $\bar{X}^u$ is unavailable due to privacy regulations, in such cases, $\mathcal{M}_t$ is still a powerful model benefiting from its deep neural structure and contextual input, and is qualified to act as a teacher model. We will validate this argument through experiments.

4.3 Knowledge Distillation Loss

It is essential to develop a proper knowledge distillation loss function $\mathcal{L}_{KD}$ to ensure the knowledge distillation quality. There is a lot of work developing task-specific KD loss functions. For example, for multi-class classification tasks, it is common to transfer knowledge from the teacher model to the student by minimizing a loss function in which the target is the distribution of class probabilities predicted by the teacher model [10]. However, there is rare work focusing on recommendation tasks under the knowledge distillation framework. One of the most recent and relevant work is Ranking Distillation [32] that designs a distillation loss to make the student model to additionally learn the reliable information from the top M items on the ranking list given by the well-trained teacher model. However, this method only accounts for a few highly-ranked items generated by the teacher model as augmented guidelines, and the low-ranking items are simply discarded. Considering that significant computing resources are consumed to generate this valuable list, we should also make full use of those low-ranked items.

To this end, we implement $\mathcal{L}_{KD}$ to be a pair-wise ranking loss instead of point-wise, and one key of designing an effective pair-wise loss function in recommender systems is to choose appropriate negative samples [34]. The most straightforward and popular method is random sampling, but it is too ordinary to ensure that the random samples are true negative as they might have not yet been exposed to the user and could be positive, which violates the assumption of the pair-wise ranking loss. Instead, we propose to take the low-ranked items calculated by the teacher model as negative samples, and such negative samples are essentially safer and more suitable since they are already evaluated by the "experienced" teacher. For

a given sequence, the top-K and bottom-K recommendations generated by the well-trained teacher from a pool with M candidates can be regarded as additional positive and negative samples passed to the student model. Through the ranking list, the teacher model can implicitly transfer the knowledge to the student model, which guides the discovery of correlations between the check-in sequence and the next POI. It should be noted that K is considered as a hyper-parameter used to measure the confidence of the expertise of the teacher model, and a larger K means more possible knowledge is transferred to the student model. Based on these justifications, we formulate this distillation loss $\mathcal{L}_{KD}$ as:

$$\mathcal{L}_{KD} = -\frac{1}{K}\sum_{i=1}^{K} w_i \log\left(\sigma\left(s_i - s_{i'}\right)\right) \tag{22}$$

where s_i and $s_{i'}$ are respectively the scores given by the student model for the i-th and the i-th last position in the ranking list output from the teacher, and w_i is the confidence level for the knowledge to be learned from the ranking list, and we follow the simple but effective *weighting by position importance* weighting scheme proposed in [32] to measure it. In this scheme, items in the topmost and bottommost positions are naturally considered to be closer to the ground-truths, so the weight w_i should be strongly related to the rank positions. We adopt a flexible parameterized geometric distribution to make the weights position-dependent:

$$w_i \propto e^{-i/\beta} \tag{23}$$

where the hyper-parameter β controls the emphasis to be placed on the top/bottom items.

Before distributing and deploying LLRec to mobile devices to start serving users, we train it with a two-step offline process on the servers. (I) We fully train the teacher model $\mathcal{M}_t$ based on all available contextual sequences and user check-in histories w.r.t. the pair-wise loss. (II) For a given plain activity sequence X with $T-1$ check-ins, we build a contextual sequence $\bar{X}$ and a user's long-term sequence $\bar{X}^u$ corresponding by retrieving related data from the server storage. Then, for the next time step T, the well-trained teacher model generates a top-K recommendation list L based on all activities in $\bar{X}$ and $\bar{X}^u$ prior to this time step. Finally, LLRec is trained to predict the ground-truth POI at T w.r.t. the ranking loss $\mathcal{L}_{SL}$ based on all activities prior to T in X. Meanwhile, LLRec is also trained w.r.t. the distillation loss $\mathcal{L}_{KD}$ based on the exemplary top-K recommendation list L.

5 EXPERIMENTS

In this section, we empirically study our proposals.

5.1 Datasets & Evaluation Protocols

First, we introduce the basic experimental setups.

Datasets We use the real-world datasets [19] collected from two popular location-based social network platforms, namely Weeplaces and Gowalla. The Weeplaces dataset has 7,658,368 check-ins from 15,799 users over 971,309 locations. The Gowalla dataset has 36,001,959 check-ins from 319,063 users over 2,844,076 locations. For both datasets, we only keep users having more than 10 check-ins, which is a common practice adopted in this domain [2, 16, 48]. After pre-processing, the Weeplaces and Gowalla datasets contain 644,244 and 1,979,826 locations, respectively.

We partition the ordered user check-in activities by one day, which is a natural segmentation granularity [13], to obtain plain check-in activity sequences X, and we further consider the location's textual category description as the contextual feature and construct the contextual check-in activity sequence $\bar{X}$ and $\bar{X}^u$.

Evaluation Protocols Following previous work [34, 35, 42], we adopt the widely-used *leave-one-out* protocol. For each check-in sequence, we select the most recent check-in POI as the ground truth for test, and utilize the remaining for training. For each ground truth POI, we pair it with 100 randomly selected POIs that this user has never visited. Then, a recommendation model generates a ranked list for those 101 POIs based on its similarity scores, where we expect the ground-truth POI is ranked at the top. To measure the recommendation effectiveness, we employ Hit Ratio (HR@k) and Normalized Discounted Cumulative Gain (nDCG@k) on the top-k POIs of the ranked list, which are well-established in the literature [9, 42]. Intuitively, HR@k simply considers whether the ground truth has appeared among the top-k POIs while nDCG@k is a position-aware ranking metric.

5.2 Baseline Methods

We compare LLRec with the following baseline methods:

LSTM [11]: LSTM has shown its ability in handling sequential data and in capturing long-term dependencies. Therefore, it is natural to apply LSTM to the next POI recommendation problem.

ST-RNN [18]: ST-RNN incorporates the time- and distance-specific transitions into the the standard RNN transition matrix to model spatiotemporal contexts.

ST-LSTM [48]: ST-LSTM generalizes the LSTM gating mechanism with spatial and temporal gates.

TMCA [16]: This is a state-of-the-art method that leverages the attention mechanism to select the relevant check-ins and contextual features to predict the next POI.

CARA [20]: CARA leverages both check-in sequences and contextual information associated with the sequences to capture users' dynamic preferences by using the gating mechanism.

5.3 Effectiveness Analysis on Teacher Model

One basic assumption in knowledge distillation is that the teacher model is sufficiently powerful to assist training the student model. Therefore, before analyzing the lightweight LLRec model, we first conduct experiments for the teacher model $\mathcal{M}_t$ by comparing it with the baseline models. Notably, we also develop two degraded variants of $\mathcal{M}_t$ based on its components $\mathcal{M}_t^R$ and $\mathcal{M}_t^A$ alone. In short, as $\mathcal{M}_t^R$ only considers the short-term spatiotemporal transitions and $\mathcal{M}_t^A$ only models the long-term user check-in behaviors using the attention mechanism, we are able to quantify the contributions from our proposed long- and short-term components. To simulate real-world scenarios and be consistent with the model settings, the short-term contextual check-in activity sequences $\bar{X}$ are used as the input for $\mathcal{M}_t^R$ and all baselines. $\mathcal{M}_t^A$ is fed with users' contextual activity sequences $\bar{X}^u$, and the full teacher model $\mathcal{M}_t$ takes both $\bar{X}$ and $\bar{X}^u$ as its input.

The experimental results are shown in Table 3. From the results, we draw the following observations. Firstly, on both datasets, the full teacher model $\mathcal{M}_t$ consistently outperforms on both evaluation

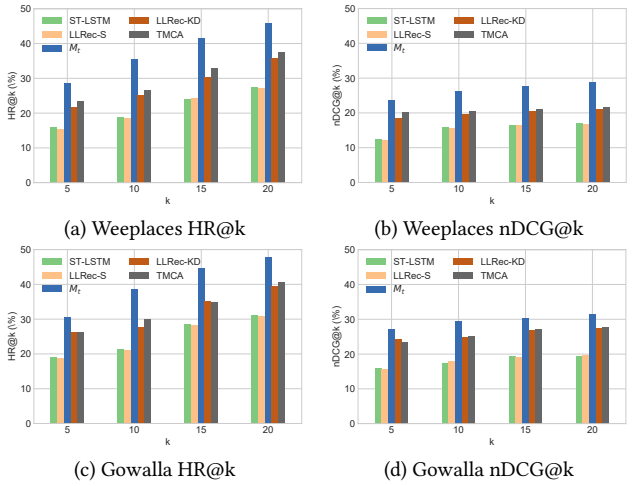
Table 3: Performance of eight models in the next POI recommendation task on two datasets

Dataset	Weeplaces								Gowalla							
Metric	HR@k (%)				nDCG@k (%)				HR@k (%)				nDCG@k (%)			
k	5	10	15	20	5	10	15	20	5	10	15	20	5	10	15	20
LSTM	16.57	19.82	21.65	24.22	14.23	17.43	18.91	19.36	18.69	22.72	24.66	26.81	15.95	20.79	22.00	22.56
ST-RNN	19.46	22.80	27.38	31.76	16.64	19.81	20.77	21.15	22.26	25.18	31.87	34.26	18.30	22.89	23.05	23.53
ST-LSTM	21.76	24.44	30.91	35.81	20.24	21.97	23.58	23.97	23.99	27.19	34.83	36.14	22.84	23.22	26.07	26.71
CARA	22.47	25.70	31.53	36.52	19.51	22.12	23.2	23.82	24.69	28.64	34.51	40.62	21.90	22.12	25.61	26.44
TMCA	23.47	26.47	32.90	37.48	19.67	22.92	23.84	24.35	26.20	29.97	34.98	40.72	23.49	25.07	27.11	27.76
$\mathcal{M}_t^R$	20.22	24.95	29.44	32.96	20.14	20.51	21.23	21.83	22.71	26.26	35.17	36.81	20.03	23.79	24.7	25.38
$\mathcal{M}_t^A$	24.84	27.90	32.64	36.28	21.69	22.98	24.06	24.85	26.48	29.84	34.74	38.65	24.63	25.20	26.53	27.82
$\mathcal{M}_t$	28.56	35.42	41.40	45.94	23.80	26.20	27.74	28.90	30.49	38.74	44.57	47.81	27.15	29.59	30.24	31.45

metrics by a large margin, and the performance differences between $\mathcal{M}_t$ and other baselines are statistically significant ($p < 0.01$). This shows that $\mathcal{M}_t$ is fully capable of serving as a powerful guiding teacher in the knowledge distillation framework for next POI recommendation. Secondly, both variants $\mathcal{M}_t^R$ and $\mathcal{M}_t^A$ provide lower recommendation accuracy in comparison to the full model, proving the advantage of jointly modeling short-term spatiotemporal sequential influences and long-term user preferences. Thirdly, the teacher's recurrent component $\mathcal{M}_t^R$ alone achieves similar performance as the state-of-the-art next POI recommendation models TMCA and CARA, and even outperforms TMCA in some settings (i.e., nDCG@5 on Weeplaces and HR@15 on Gowalla). Also, $\mathcal{M}_t^R$ constantly outperforms ST-RNN that has no gating mechanism and LSTM that does not consider spatiotemporal influences, which verifies the benefits brought by the gating mechanism to capture spatiotemporal transitions in modeling sequential check-in patterns. As the privacy regulations may prevent recommendation models from accessing the sensitive contextual data (e.g., location), $\mathcal{M}_t^R$ can effectively guarantee high-quality recommendation performance in such circumstances. The forth observation is that $\mathcal{M}_t^R$ has comparable performance as ST-LSTM, reflected by better HR@10 on Weeplaces and higher HR@15, HR@15 and nDCG@10 on Gowalla. This demonstrates that adopting FastGRNN as our building block can yield highly competitive performance as LSTM-based models, but with a substantially lower model complexity.

5.4 Effectiveness Analysis on LLRec

Based on the overall effectiveness in Table 3, we now compare the recommendation performance of LLRec with the best baselines. Specifically, we compare LLRec with TMCA and the full teacher model $\mathcal{M}_t$, which perform the best against other baselines. Also, as LLRec only takes the plain check-in sequence as input, we further select ST-LSTM, which can be easily converted into a context-free model, as an additional competitor. In our experiments, we use LLRec-S to denote the student LLRec model trained solely with the BPR loss in Eq 17 and the plain check-in sequences, and use LLRec-KD to denote the LLRec model trained with the knowledge distillation-based teacher model via the loss in Eq 16. Note that LLRec-KD is the default version of LLRec to be finally deployed on mobile devices. We plot the recommendation results in Figure 1. Like LLRec-S, ST-LSTM is modified to only consume the plain check-in activity sequences X.



(a) Weeplaces HR@k (b) Weeplaces nDCG@k

(c) Gowalla HR@k (d) Gowalla nDCG@k

Figure 1: Performance of LLRec

From the results, we firstly see that both variants of LLRec have comparable performance as ST-LSTM, proving, again, that our proposed method is an ideal alternative to LSTM-based approaches and can achieve high recommendation quality with a much lighter architecture. Secondly, LLRec-S falls behind $\mathcal{M}_t$ in all test cases, and this is in line with real-life application scenarios because the auxiliary contextual information can indeed help improve the effectiveness of next POI recommendation. Since LLRec-S can be viewed as a completely offline model trained with locally generated user data, the rich contextual data is beyond its reach and thus is not taken into consideration, which results in decreased performance. Thirdly, it is worth mentioning that the performance of LLRec-KD increases from LLRec-S significantly under all settings, and the performance differences between LLRec-KD and TMCA are minimal. Specifically, for Weeplaces, the average improvements of LLRec-KD over LLRec-S are 7% and in HR and 5% in nDCG respectively, and similar patterns can be spotted on Gowalla where HR and nDCG increase by an average of 8%. This confirms that the knowledge distillation training is highly effective in improving the performance of LLRec due to the additional knowledge transferred from the powerful teacher model.

Table 4: HR@10 (in percentage) of four LLRec models on Weeplaces using different tensor shapes

Model	Embedding Shape	HR@10	Size	Compr.
LLRec-1	644244×128	18.91	82,463,232	1
LLRec-2	$(444, 1451) \times (16, 8)$	18.46	299,392	275
LLRec-3	$(12,37,1451) \times (8, 4, 4)$	**18.60**	132,288	623
LLRec-4	$(3,4,37,1451) \times (4, 4, 4, 2)$	17.47	88,608	930

Table 5: HR@10 (in percentage) of LLRec-3 on Weeplaces using different TT-ranks and embedding factorization

TT-rank	Embedding Shape	HR@10	Size	Compr.
8	$(12,37,1451) \times (8, 4, 4)$	16.52	56,672	1455
	$(12,37,1451) \times (16, 4, 2)$	15.27	34,224	2409
16	$(12,37,1451) \times (8, 4, 4)$	18.60	132,288	623
	$(12,37,1451) \times (16, 4, 2)$	17.18	87,392	943
32	$(12,37,1451) \times (8, 4, 4)$	19.35	340,352	242
	$(12,37,1451) \times (16, 4, 2)$	18.77	250,560	329

5.5 Effects of Model Compression

In this study, we first evaluate the effects of the proposed tensor-train compression technique for the embedding layer. Specifically, we compare the prediction qualities of the uncompressed LLRec (LLRec-1) with three compressed LLRec versions named LLRec-2, LLRec-3 and LLRec-4 using varied tensor dimensions. The default version of LLRec (i.e., LLRec-KD) is tested in this section. The TT-rank and embedding dimension are respectively set to 16 and 128 for the experiments. In Table 4, we report the the compression rates, embedding layer sizes, compression rates and the prediction accuracy on the Weeplaces dataset in terms of HR@10. where the compression rate is measured by Eq 15. In the table, we find that heavily compressed models can still perform closely to the uncompressed LLRec-1 even with much fewer parameters. For example, the performance of LLRec-3 drops very slightly compared with LLRec-1 (from 18.91% to 18.6%), but the parameter size is remarkably reduced by more than 600 times. Another finding is that LLRec-3 performs better than LLRec-2 even with higher compression rate, which implies that an appropriate low-rank tensor structure for the embedding matrix can be viewed as a special form of regularization, thus contributing to better performance.

To discover the relation among TT-rank, compression rates and prediction accuracy, we further experiment with three different TT-ranks for the TT-format. As LLRec-3 achieves the best performance among the compressed models, we follow the settings of LLRec-3 (i.e., the tensor dimension is 3 and the embedding dimension is 128) and vary the size of TT-ranks and embedding dimensions in this test. In Table 5, we found that the compression rates are extremely sensitive to the TT-rank. For instance, when we double the TT-rank, the decrease of the compression rate is more than 200%. In addition, the aggressive compression rate from lower TT-ranks dose not necessarily come with the dramatic losses of recommendation accuracy. For example, HR@10 only decreases about 1% if we reduce the TT-rank from 32 to 16. Therefore, TT-rank is a critical hyper-parameter for compressing the model to fit in the mobile devices, and also plays an important role in balancing the model size and performance. It is worth mentioning that, in our experimental environments, by saving the models with the default serialization functions from Pytorch [25], the eventual model size is only 1.7 Mb for LLRec-3, which can be considered a truly mobile-friendly recommendation model in contrast to the 300 Mb LLRec-1.

In conclusion, by carefully choosing the hyper-parameters, the LLRec can be easily transmitted via the network and deployed on the resource-constrained mobile devices without excessively sacrificing the recommendation quality.

Table 6: Prediction time (in ms) on the mobile device

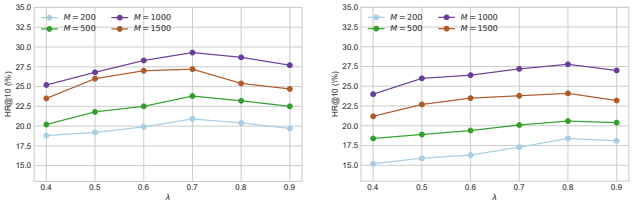
Dataset	# of Cand.	ST-RNN	ST-LSTM	LLRec
Weeplaces	500	5,633	40,855	633
	200	2,566	24,064	279
Gowalla	500	6,378	31,565	760
	200	3,293	20,729	317

5.6 Model Efficiency

We also study our model's efficiency when generating a recommendation list on mobile devices. Specifically, we deploy the well-trained ST-RNN, ST-LSTM and the default LLRec to a smartphone (Google Pixel 3) with 4 Gigabytes (GB) RAM and Qualcomm Snapdragon 670 processor. We select ST-RNN and ST-LSTM because they are compatible with the context-free recommendation setting on mobile devices, while TMCA and CARA are invalid with such simplified input data. We then simulate the real-world recommendation scenario by assigning them a short-term check-in activity sequence without contextual features and measure the running time for generating a top-10 recommendation list from 200 and 500 candidates. We conduct experiments on both Weeplaces and Gowalla, and report the results in Table 6. Clearly, for both datasets, LLRec is 40x - 65x faster than ST-LSTM and is 8x - 10x faster than ST-RNN. The consistently short running time enables LLRec to locally perform real-time next POI recommendations on users' mobile devices.

5.7 Effects of Knowledge Distillation

Finally, we explore the effects of knowledge distillation where we test HR@10 of LLRec on both datasets w.r.t. different settings of two critical hyper-parameters in our KD method, i.e., the number of candidates M the teacher model considers to generate positive-negative POI pairs, and λ to control the weights of the student loss $\mathcal{L}_{SL}$ and the distillation loss $\mathcal{L}_{KD}$. We can draw two conclusions from the results shown in Figure 2. Firstly, it is important to balance the contributions of knowledge from the ground-truth dataset and the teacher model's guidance to train the student model. In essence, the student still needs to learn most of the knowledge (about 70% - 80%) itself from the ground-truth dataset, and this also aligns with the intuition. The other conclusion is that we need to set a proper number of candidate POIs for the teacher to evaluate. If M is too small, it may not contain enough true positive and true negative POIs, and if M is too large, the probability that the teacher returns a false positive or a false negative increases. As indicated in Figure 2, both cases have the risk of leading to sub-optimal parameters.



(a) Weeplaces (b) Gowalla

Figure 2: Effects of M and λ in the knowledge distillation

6 RELATED WORK

In this section, we review the related work in the related areas of POI recommendation and deep neural networks compression.

6.1 POI Recommendation

The next POI recommendation recommendation research is highly related to the general POI recommendation. Conventional POI recommender systems inherit the collaborative filtering technique that exploits the relationship among users, POIs and contextual features, such as [17, 19, 41]. IRenMF [19] incorporates neighborhood characteristics into the learning of latent factors of users and locations, and GeoMF [17] extends the factorization score with the influence of geographical region. Recently, neural network techniques have been introduced to develop POI recommender systems that achieve the state-of-art performance [38, 43, 49]. Xie et al. [38] proposed to jointly learn the POI-POI, POI-region, POI-time and POI-word bipartite graphs and obtain the POI embedding. In [49], the POI representation is learned under different temporal states in order to capture various temporal characteristics on different days. Recent years witnessed the increasing research interest towards next POI recommendation. Early work to address this problem is based on Probabilistic Graphical Models [33, 44, 45] and Markov chains [4, 37]. For example, the tensor-based FPMC-LR [4] including the personalized Markov chains and the localized region constraint was proposed to recommend a successive personalized POI. The latest trend is to exploit recurrent neural network good at modeling sequential data to tackle the next POI recommendation problem. In [18], a time-specific and distance-specific transition matrix in RNN to model local spatiotemporal contexts was introduced. [13] and [48] integrate spatial and temporal gating mechanism into LSTM to incorporate spatiotemporal information for prediction. TMCA [16] leverages the attention mechanism to select the relevant check-ins and contextual features to predict the next POI.

However, those models serve the users' recommendation needs in a cloud-based paradigm, namely, they are trained and deployed on resource-rich cloud servers, usually ignoring the model size and running efficiency. On the contrary, our proposed small-scale LLRec can run on local resource-constrained mobile devices to provide high-quality and timely next POI recommendation.

6.2 Deep Neural Networks Compression

It is an increasingly important research topic to compress deep neural networks to balance their effectiveness and efficiency. There are three major lines of work in the literature: pruning, quantization and knowledge distillation.

Network pruning methods [3, 8, 30] aim to reduce the complexity and the over-fitting in networks by removing parameters that are not necessary or have little impact for inference. The recent trend in this line is to train compact neural networks with sparsity constraints. [51] introduces a group-sparse regularization on neurons during the training stage to learn compact CNNs with reduced filters. Most of the pruning methods allow significant parameter reduction without change or significant drop in accuracy, but the major drawback is that they suffer from expensive time cost since the network pruning requires more iterations to converge. The second line is quantization methods [1, 5, 7] that reduce the number of bits required to represent every weight, decreasing the number of parameters by exploiting redundancy. Pruning and quantization are often used together to achieve a solid compression rate. Recently, knowledge distillation (KD) [10] as a model-independent learning framework gains popularity because of its flexibility. In KD, a large and cumbersome teacher model is first trained, and subsequently soft-labels obtained from the teacher model are used as training data for a smaller target student model, gaining comparable performance in the end. Research on KD techniques is three-fold: (1) knowledge distillation from outputs such as the original KD [10] and Ranking Distillation (RD) [32] (2) knowledge distillation from a single layer such as FitNext [28] and (3) knowledge distillation from multiple layers such as [40]. The knowledge distillation method in our work is inspired by RD where both the teacher and student are ranking models. RD only considers the top ranked items as soft positive feedback but ours additionally considers the bottom ranked items as soft negative feedback, and apply them to a pairwise loss. This novel extension enables the student model to learn more complex item correlations from the teacher and be in line with the nature of ranking models.

7 CONCLUSION

Contemporary next POI recommendation services are mostly cloud-based, causing problems regarding high demand on network quality, huge energy consumption, privacy concerns, etc., and we proposed a novel local recommendation paradigm as an effective and efficient solution. We first proposed a compact yet high-performance model LLRec that can be directly deployed on resource-constrained mobile devices. With the support of the lightweight FastGRNN, LLRec incorporates the critical spatiotemporal patterns with the gating mechanism, while consuming a fraction of computing resources on mobile devices as our novel tensor-train compression technique significantly reduces the parameter size in its embedding layer. As a simplified model, LLRec maintains its robustness when confronted with extremely limited data availability in mobile environments via the proposed knowledge distillation-based training framework, where the LLRec acts as a student model guided by a powerful teacher model trained with the rich contextual data on the cloud. To verify the real-world adaptivity of LLRec, we carried out extensive experiments, which fully demonstrate our model's advantageous effectiveness and efficiency in the next POI recommendation task.

ACKNOWLEDGMENTS

This work is supported by Australian Research Council (Grant No. DP190101985, DP170103954).

REFERENCES

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv* (2013).

[2] Buru Chang, Yonggyu Park, Donghyeon Park, Seongsoon Kim, and Jaewoo Kang. 2018. Content-Aware Hierarchical Point-of-Interest Embedding Model for Successive POI Recommendation. In *IJCAI*. 3301–3307.

[3] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing Neural Networks with the Hashing Trick. In *ICML*. 2285–2294.

[4] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-Interest Recommendation. In *IJCAI*.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training Deep Neural Networks with Low Precision Multiplications. *arXiv* (2014).

[6] Yann N Dauphin and Yoshua Bengio. 2013. Big Neural Networks Waste Capacity. *arXiv* (2013).

[7] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *International Conference on Machine Learning*. 1737–1746.

[8] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning Both Weights and Connections for Efficient Neural Network. In *NeurIPS*. 1135–1143.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv* (2015).

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. 2019. Tensorized Embedding Layers for Efficient Model Compression. *arXiv* (2019).

[13] Dejiang Kong and Fei Wu. 2018. HST-LSTM: A Hierarchical Spatial-Temporal Long-Short Term Memory Network for Location Prediction.. In *IJCAI*. 2341–2347.

[14] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In *NeurIPS*. 9031–9042.

[15] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-Based Recommendation. In *CIKM*. 1419–1428.

[16] Ranzhen Li, Yanyan Shen, and Yanmin Zhu. 2018. Next Point-of-Interest Recommendation with Temporal and Multi-level Context Attention. In *ICDM*. 1110–1115.

[17] 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 *KDD*. 831–840.

[18] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts. In *AAAI*.

[19] Yong Liu, Wei Wei, Aixin Sun, and Chunyan Miao. 2014. Exploiting Geographical Neighborhood Characteristics for Location Recommendation. In *CIKM*. 739–748.

[20] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. 2017. A Deep Recurrent Collaborative Filtering Framework for Venue Recommendation. In *CIKM*. 1429–1438.

[21] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. 2018. A Contextual Attention Recurrent Architecture for Context-Aware Venue Recommendation. In *SIGIR*. 555–564.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv* (2013).

[23] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing Neural Networks. In *NeurIPS*. 442–450.

[24] Ivan V Oseledets. 2011. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing* 33, 5 (2011), 2295–2317.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in Pytorch. (2017).

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.

[27] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing Personalized Markov Chains for Next-Basket Recommendation. In *WWW*. 811–820.

[28] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for Thin Deep Nets. *arXiv* (2014).

[29] Frank Seide, Gang Li, and Dong Yu. 2011. Conversational Speech Transcription Using Context-Dependent Deep Neural Networks. In *INTERSPEECH*.

[30] Suraj Srinivas and R Venkatesh Babu. 2015. Data-Free Parameter Pruning for Deep Neural Networks. *arXiv* (2015).

[31] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H Chi. 2019. Towards Neural Mixture Recommender for Long Range Dependent User Sequences. *arXiv* (2019).

[32] Jiaxi Tang and Ke Wang. 2018. Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System. In *KDD*. 2289–2298.

[33] Hao Wang, Yanmei Fu, Qinyong Wang, Hongzhi Yin, Changying Du, and Hui Xiong. 2017. A Location-Sentiment-Aware Recommender System for Both Home-Town and Out-of-Town Users. In *KDD*. 1135–1143.

[34] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural Memory Streaming Recommender Networks with Adversarial Training. In *KDD*. 2467–2475.

[35] Qinyong Wang, Hongzhi Yin, Hao Wang, Nguyen Quoc Viet Hung, Zi Huang, and Lizhen Cui. 2019. Enhancing Collaborative Filtering with Generative Augmentation. In *KDD*.

[36] Shoujin Wang, Liang Hu, Longbing Cao, Xiaoshui Huang, Defu Lian, and Wei Liu. 2018. Attention-Based Transactional Context Embedding for Next-Item Recommendation. In *AAAI*.

[37] Weiqing Wang, Hongzhi Yin, Shazia Sadiq, Ling Chen, Min Xie, and Xiaofang Zhou. 2016. SPORE: A Sequential Personalized Spatial Item Recommender System. In *ICDE*. 954–965.

[38] Min Xie, Hongzhi Yin, Hao Wang, Fanjiang Xu, Weitong Chen, and Sen Wang. 2016. Learning Graph-Based Poi Embedding for Location-Based Recommendation. In *CIKM*. 15–24.

[39] Yinchong Yang, Denis Krompass, and Volker Tresp. 2017. Tensor-Train Recurrent Neural Networks for Video Classification. In *ICML*. 3891–3900.

[40] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning. In *CVPR*. 4133–4141.

[41] Hongzhi Yin, Bin Cui, Xiaofang Zhou, Weiqing Wang, Zi Huang, and Shazia Sadiq. 2016. Joint Modeling of User Check-in Behaviors for Real-time Point-of-Interest Recommendation. *TOIS* 35, 2 (2016), 1–44.

[42] Hongzhi Yin, Qinyong Wang, Kai Zheng, Zhixu Li, Jiali Yang, and Xiaofang Zhou. 2019. Social Influence-Based Group Representation Learning for Group Recommendation. In *ICDE*. 566–577.

[43] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. 2017. Spatial-Aware Hierarchical Collaborative Deep Learning for POI Recommendation. *TKDE* 29, 11 (2017), 2537–2551.

[44] Hongzhi Yin, Xiaofang Zhou, Bin Cui, Hao Wang, Kai Zheng, and Quoc Viet Hung Nguyen. 2016. Adapting to User Interest Drift for POI Recommendation. *TKDE* 28, 10 (2016), 2566–2581.

[45] Hongzhi Yin, Xiaofang Zhou, Yingxia Shao, Hao Wang, and Shazia Sadiq. 2015. Joint Modeling of User Check-in Behaviors for Point-of-Interest Recommendation. In *CIKM*. 1631–1640.

[46] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A Dynamic Recurrent Model for Next Basket Recommendation. In *SIGIR*. 729–732.

[47] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. 2014. Sequential Click Prediction for Sponsored Search with Recurrent Neural Networks. In *AAAI*.

[48] Pengpeng Zhao, Haifeng Zhu, Yanchi Liu, Zhixu Li, Jiajie Xu, and Victor S Sheng. 2018. Where to Go Next: A Spatio-temporal LSTM model for Next POI Recommendation. *arXiv* (2018).

[49] Shenglin Zhao, Tong Zhao, Irwin King, and Michael R Lyu. 2017. Geo-Teaser: Geo-Temporal Sequential Embedding Rank for Point-of-Interest Recommendation. In *WWW*. 153–162.

[50] Shenglin Zhao, Tong Zhao, Haiqin Yang, Michael R Lyu, and Irwin King. 2016. STELLAR: Spatial-Temporal Latent Ranking for Successive Point-of-Interest Recommendation. In *AAAI*.

[51] Hao Zhou, Jose M Alvarez, and Fatih Porikli. 2016. Less Is More: Towards Compact CNNs. In *ECCV*. Springer, 662–677.