RC-TL: Reinforcement Convolutional Transfer Learning for Large-scale Trajectory Prediction

Negar Emami, Lucas Pacheco, Antonio Di Maio, Torsten Braun

Institute of Computer Science, University of Bern, Switzerland
Email: {negar.emami, lucas.pacheco , antonio.dimaio, torsten.braun}@inf.unibe.ch

***Abstract*—Anticipating future locations of mobile users plays a pivotal role in intelligent services supporting mobile networks. Predicting user trajectories is a crucial task not only from the perspective of facilitating smart cities but also of significant importance in network management, such as handover optimization, service migration, and the caching of services in a mobile and edge-computing network. Convolutional Neural Networks (CNNs) have proven to be successful to tackle the forecasting of mobile users' future locations. However, designing effective CNN architectures is challenging due to their large hyper-parameter space. Reinforcement Learning (RL)-based Neural Architecture Search (NAS) mechanisms have been proposed to optimize the neural network design process, but they are computationally expensive and they have not been used to predict user mobility. In large urban scenarios, the rate at which mobility information is generated makes it a challenge to optimize, train, and maintain prediction models for individual users. However, considering that user trajectories are not independent, a common trajectory-prediction model can be built and shared among a set of users characterized by similar mobility features. In the present work, we introduce Reinforcement Convolutional Transfer Learning (RC-TL), a CNN-based trajectory-prediction system that clusters users with similar trajectories, dedicates a single RL agent per cluster to optimize a CNN neural architecture, trains one model per cluster using the data of a small user subset, and transfers it to the other users in the cluster. Experimental results on a large-scale dataset show that our proposed RL-based CNN achieves up to 12% higher trajectory-prediction accuracy, with no training speed reduction, over other state-of-the-art approaches on a large-scale, real-world mobility dataset. Moreover, RC-TL's clustering strategy saves up to 90% of the computational resources needed for training compared to single-user models, in exchange for a 3% accuracy reduction.**

Keywords: Trajectory Prediction, Convolutional Neural Network, Reinforcement Learning, Clustering, Transfer Learning.

I. Introduction

Next-generation networks must be self-organized, efficient, and cost-effective. While 5G networks are already being deployed worldwide, Beyond 5G (B5G) and its successors must seamlessly manage user mobility providing high Quality of Experience (QoE) for their consumers [1]. User mobility is a defining characteristic of modern networks, and is responsible for management events such as Handover (HO) and service migration. While mobile networks have traditionally adopted reactive strategies, next-generation networks must perform mobility management in a proactive manner in order to provide high-Quality of Service (QoS) mobility for their users. Such proactive mobility management strategies rely on the usage of Artificial Intelligence (AI) to forecast and handle user mobility. Adaptive and anticipatory network management enables the operator to allocate resources, optimize the costs of the network, and improve user experience. For instance, trajectory prediction enables proactive mechanisms so that handover signaling can be done before the mobile user's arrival at a target base station to guarantee communication continuity. Furthermore, based on mobility knowledge, various heterogeneous services can be offloaded from the cloud to servers located at the edge of the network closer to end-users. Service migration is an integral approach for keeping services close to users even as they move to guarantee QoS levels for the applications they consume [2]. On the other hand, smart cities rely on mobility information to support Intelligent Transportation Services (ITS) such as safety technologies for the development of fully autonomous vehicles, collision avoidance, efficient road traffic management, navigation, and route recommendation systems [3], [4]. Therefore, designing an accurate, optimized, and robust mobility predictor is a critical task for both management and consumer services.

Machine Learning (ML) and Artificial Neural Network (ANN) models have proven to be highly effective in predicting mobility by analyzing users' historical location data and learning moving patterns to forecast future user locations. In particular, Long Short Term Memory (LSTM) models are already mature in the field of time-series prediction, and a few recent studies show the effectiveness of CNNs to tackle this problem. While CNNs are not commonly applied to learn time-series data such as user mobility information, they have shown significant potential due to their ability to encode patterns in their convolutional kernels. Furthermore, CNN allows learning parallelization and requires limited computation to perform the prediction task.

However, LSTM and CNN models are characterized by a large set of *hyper-parameters* (e.g., activation functions, diversity in type and number of neural layers, dropout values, and units) that are commonly determined by a human expert

978-1-6654-0601-7/22/$31.00 © 2022 IEEE

heuristically [5], [6], [7]. Some works optimize the neural architecture using RL, a self-learning model that finds the most appropriate neural network architecture for a given dataset by rewarding architectures that lead to high prediction accuracy. Searching for an optimal neural network architecture for each user using RL can be computationally expensive in large systems. Some studies show that clustering users with similar characteristics and training a single model for all the users in the cluster can reduce the computation needed for training [8], [9]. However, no studies have investigated the potential of user clustering to reduce the computational requirements for RL-based NAS, which is one of the goals of this study.

In this work, we propose RC-TL, an efficient CNN-based trajectory predictor that automates and personalizes neural network architecture design at the cluster level. RC-TL 's novelty is threefold. First, it applies for the first time RL-CNN to predict user mobility. Second, it groups users with similar trajectories and builds one RL-CNN model per cluster based on a few representative users, reducing the computation needed to train the model. Finally, it transfers the model trained on the cluster's set of representative users to the other members of that cluster using Transfer Learning (TL).

The rest of this paper is organized as follows. Section II presents the related works. Section III describes RC-TL's operation. Section IV evaluates RC-TL's performance. Finally, Section V concludes the paper, summarizes the contributions of this work, and indicates some future avenues.

II. Related Works

In recent years, AI algorithms have replaced conventional statistical models to analyze mobility data and predict future trajectories of mobile users, e.g., pedestrians and vehicles [10], [11], [12]. In the literature, the problem of trajectory prediction has been extensively tackled with Recurrent Neural Networks (RNNs) [5], and their variants, such as LSTM [6] as successful Deep Learning (DL) models designed for time-series prediction. Despite the success of RNNs and LSTMs, they face a slow training process, as they must process the input data in sequential order. In contrast, CNN's are other powerful DL tools that can concentrate on sequential data from a hierarchical perspective processing the data as a whole and, thus, can become a suitable alternative for RNNs. The predictors presented by Nikhil et al. [13] and Zamboni et al. [7] is from the limited existing works that apply CNNs for the trajectory prediction field.

Although all aforementioned ANN-based works successfully predict trajectories, they design the neural architectures following human-expert-based heuristics, which is an error-prone and time-consuming process. Furthermore, they apply the same neural architecture to every user dataset, assuming everyone has similar mobility behaviors. Several search methods have been proposed to automate the neural architecture design process given a dataset in other fields than mobility prediction. Elsken et al. [14] surveys existing Neural Architecture Search (NAS) works where hyper-parameter optimization can be formulated within the scope of an RL algorithm as a more efficient search technique concerning the naive search techniques such as grid search, random search [15], and Bayesian search [16]. Most of the Neural Architecture Search (NAS) works leverage RL to find the best architecture for image classification tasks, where defining well-suited search spaces is relatively easy due to human experience and the existence of several manually-designed models [17], [18], [19]. However, the potential of NAS methods in less explored domains is still unclear [14]. Therefore we propose and study a RL-based NAS method in the field of trajectory prediction.

Computationally light NAS methods have not been studied in the literature [14]. RL for NAS is computationally lighter than Grid Search (GS) methods and can achieve better accuracy than random search methods, but is still computationally expensive. In a real network scenario with thousands of mobile users' trajectories, training an RL agent per each user dataset to optimize their neural network architecture would be impossible. Multiple users in an urban area might partially follow similar trajectories during a specific period of the day by chance or by groups' intentions [20]. Therefore, we propose to cluster similar trajectory users and train a single RL agent for each cluster of users instead of for each user to reduce computational requirements in large-scale networks.

In the literature, some relevant trajectory prediction works have suggested clustering similar users. Nevertheless, their approaches have many shortcomings and are not within the scope of NAS. For example, the recently proposed model in [8] clusters similar trajectory users and then feeds only parts of complete trajectories (referred to as partial trajectories) within a cluster to an LSTM. However, their approach faces three problems. First, training only a part of trajectories disregards long-term spatio-temporal dependencies information. Besides, training partial trajectories of all users within all clusters still requires a considerable amount of computational resources. Moreover, Shrivastava et al. [8] apply the same heuristically-designed LSTM architecture to all clusters, which does not guarantee the optimal prediction performance. Alternatively, Sung et al. [9] cluster similar users and then aggregate all user data within a cluster by averaging them to a single data sequence and feeding it to a Markov-based predictor. This model also suffers from three shortcomings. Primarily, averaging all users' data discards useful spatio-temporal information. Furthermore, their model needs access to all users' data for the aggregation, which is expensive in terms of communication and computation. Finally, Markov-based predictors are much less potent than ANNs and cannot guarantee optimal performance. In this direction, our proposed RC-TL addresses all the shortcomings mentioned above by bringing a combination of NAS and TL to the cluster-level mobility prediction. RC-TL offers an accurate yet computationally efficient trajectory

TABLE I
COMPARISON OF THE STATE-OF-THE-ART TRAJECTORY PREDICTORS

Trajectory Predictor	Model	ANN	RL	Clustering	TL
Zhang et al. [5]	RNN	✓			
Phillips et al. [6]	LSTM	✓			
Nikhil et al. [13]	CNN	✓			
Shrivastava et al. [8]	LSTM	✓		✓	
Sung et al. [9]	Markov			✓	
RC-TL	CNN	✓	✓	✓	✓

prediction without the requirement of processing all users' data.

Table I compares our solution to existing state-of-the-art pedestrian or vehicular trajectory predictors in terms of the techniques used to achieve the mobility prediction. We can observe that the majority of works use some form of ANN as a predictor. However, RC-TL is the only compared work that employs a Reinforcement Learning-designed ANN together with Transfer Learning and Clustering techniques to achieve higher accuracy at comparable or lower computation costs.

III. MOBILE USER TRAJECTORY PREDICTION

A. Scenario

In our envisioned scenario, a set U of n mobile network users is free to move in an area in which a set S of base stations, which make up the cellular network, are deployed. Every time a mobile user enters a base station's communication range, they initiate a handshake procedure, after which the user is *connected* to that base station and added to a list of connected users. Similarly, when a mobile user exits a base station's communication range, the base station detects it and then *disconnects* the user by removing it from the list of connected users. Each base station maintains a list of timestamps at which each user has connected to and disconnected from it. Periodically, each base station sends this information to a logically centralized data repository for information processing. The data from all base stations can be aggregated by user, obtaining one *trajectory* per user in the system. We define the trajectory of the user u as a set of couples $T_u = \{(b_u(1), t_u(1)), \ldots, (b_u(m_u), t_u(m_u))\}$, where b_u is the base station ID to which the user u connected, t_u is the timestamp at which the user u connected to that base station, and m_u is the total number of data entries for user u. We define $\Theta = \{T_1, \ldots, T_n\}$ as the set of all user trajectories. With this information, the centralized system can compute the trajectory-prediction models for each user and send them the parameters via the cellular network to use them locally for predictions.

From a design perspective, the choices in the system are informed by the urban computing nature of the problem. Users often follow the same trajectories given by the city's public transport routes and economic features.

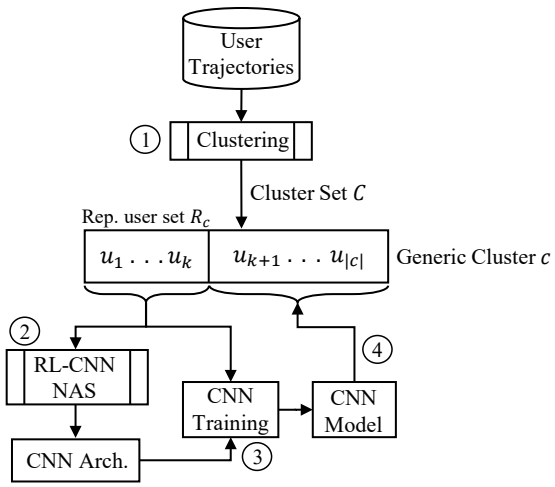

Fig. 1. RC-TL architecture. The circled numbers indicate the (1) clustering, (2) RL-CNN neural architecture search, (3) CNN training, and (4) transfer learning steps of the system.

B. Architecture

The RC-TL system is designed to optimize the prediction accuracy and the computational resource utilization at the same time. Many existing similar works aggregate all varieties of user datasets within a single neural network, which is not an optimal attitude. In this direction, to increase the average accuracy across various users, personalizing neural networks tailored to different user datasets can be an adequate approach. While this provides sufficient personalization for each user to consider different statistical features, it cannot be scaled for a large number of users. Inspired by these shortcomings, we design the RC-TL system taking into account clusters of users with similar mobility features to be learned and generalized. More precisely, RC-TL clusters similar trajectory users and trains a single RL-CNN per cluster based on a few users' data, significantly saving computation resources necessary to find the optimal architecture for such users. RC-TL provides an inter-cluster personalized yet intra-cluster generalized model.

The RC-TL system operates in a set of sequential steps, briefly described hereafter and detailed in the following subsections. Figure 1 represents the system architecture and data flow. The system's input is the database containing the user trajectories, and the system's output is one trained CNN model for each user in the scenario. The RC-TL 's goal is to reduce the computational resources required to provide a CNN model for each user while keeping high prediction accuracy.

The first step of the RC-TL architecture is the clustering, which retrieves user mobility information from a logically centralized database and groups users with similar trajectories in a set of disjoint clusters according to the Longest Common Sub-Sequence (LCSS) similarity measure. For each cluster, this step also selects a subset of *representative users* who are associated with high-quality data by evaluating their data's

coherence. The second step of the RC-TL architecture is the RL-CNN neural network architecture search, which can be run in parallel for each cluster. This step takes in input the trajectories of the cluster's representative users and uses an RL agent to find a set of hyper-parameters defining a CNN architecture that maximizes the trajectory-prediction accuracy for the representative users. After the agent selects the best CNN architecture, RC-TL trains it using the representative users' data and obtains the model's parameters in output. The fourth and last step of the RC-TL architecture is transfer learning, which takes into input the trained model's parameters and uses them to build a model for each of the other non-representative users of the cluster.

C. User Trajectory Clustering and Reference Users Selection

In real-world scenarios, the number of mobile network users can be so large that training an individual RL-CNN trajectory predictor for each user is unfeasible due to the infrastructure's limited computing and storage resources. For this reason, we designed RC-TL to cluster users with similar trajectories using the LCSS algorithm [21]. Contrary to traditional Euclidean distance, the LCSS-based distance can be computed between trajectories made of a different number of data points (t_i, b_i). RC-TL computes the LCSS distance $\gamma_{i,j} \in [0,1]$ between every pair of trajectories $(T_i, T_j) \in \Theta^2$ in the dataset, and populates a symmetric distance matrix $\Gamma = (1-\gamma_{i,j}) \in [0,1]^{n \times n}$ that represents how different the trajectories are to one another. At this point, the system applies an unsupervised clustering algorithm to group the trajectories in a set $C = \{c_1, \ldots, c_q\}$ of disjoint clusters $c_i \in \Theta, \forall i \in \{1, \ldots, q\}$, where $c_i \cap c_j = \emptyset, \forall i, j \in \{1, \ldots, q\}, i \neq j$. Clustering algorithms are designed to determine the optimal number of clusters and their members to minimize the average intra-cluster distance and maximize the inter-cluster distance. In other words, a suitable clustering algorithm must be designed so that all trajectories in a cluster have a small distance, whereas the trajectories belonging to different clusters have considerable distance. Therefore, any clustering algorithm whose geometry is based on distances between points, to compare pairwise the distances between locations within trajectories, can be a proper candidate, e.g., Birch, DBSCAN, K-Means, Mean-Shift, Ward, and Optics. Our proposed RC-TL system uses the Birch clustering algorithm, but any other clustering algorithm, such as K-means and Ward, can be used at this step [22]. The three clustering algorithms (K-Means, Ward, and Birch) are suggested as the design choices due to their out-performance concerning the other clustering methods in terms of accuracy and resource utilization.

For each user cluster, the system selects a set of *reference users* that produced highly-descriptive mobility data, evaluated through a trajectory-regularity metric detailed hereafter. Let us define $|T_j|$ as the *length* of trajectory T_j. Let us define D_j as the number of unique base stations that appear in trajectory T_j. Experience shows that users who produce more data samples while connecting to fewer base stations display more regular mobility patterns and hence allow RC-TL to achieve better trajectory-prediction accuracy. Therefore, we propose the *regularity ratio* for the j-th user as $\rho_j = |T_j|/D_j \in [1, +\infty)$, which is used as a score to estimate the quality of the data associated with each user. Users with a higher regularity ratio have visited a limited set of base stations multiple times, meaning that it is easier for a neural network to infer a periodic behavior from such users compared to users with a low regularity ratio. The set R_c of reference users for a generic cluster c is selected as the set of the $k = |R_c| << |c|$ users in the cluster with the highest regularity ratios k is determined by a grid search that optimizes prediction accuracy.

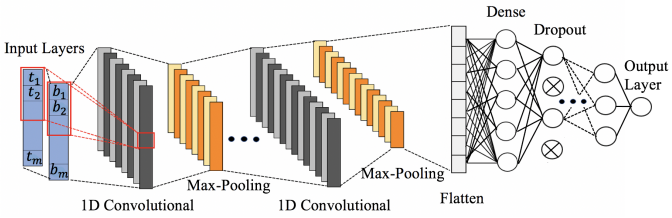

Fig. 2. Structure of the generic 1D-CNN built and trained by RC-TL.

D. One-dimensional Convolutional Neural Networks

A 1D-CNN is a sparse feed-forward ANN that can "learn" the associations between a one-dimensional *input* x and an output y. In the present work, we convert a time-series problem into a supervised learning problem, defining the input as two one-dimensional arrays (i.e., the user trajectories) and the output as the sequence of visited base stations. The process through which a CNN learns the associations between input and output is called *training*, and could be time-consuming. For this reason, we impose a limit on the number of training epochs and employ an *early stopping* training method to allow the training to end sooner if no significant accuracy improvement is made. In particular, we define Δ as the accuracy improvement threshold and the *patience* as the number of epochs that can elapse without an accuracy improvement higher than the threshold.

A CNN is composed of *layers*, each with a different purpose. In this work, we consider CNNs that can contain convolutional, max-pooling, flatten, dense, and dropout layers. Figure 2 shows the generic CNN considered in this work. A convolutional layer applies the convolution operator between the input data x and a set of *kernels* $\kappa_i \in \mathbb{R}^s$, producing a set of *feature maps* $f_i = x * \kappa_i$, where s is a fixed kernel size. The goal of the convolutional layers is to detect the presence of spatio-temporal patterns in the input data. The size and the number of the different used kernels are decided at design time, whereas the numerical values of the kernels κ_i are computed during the model training process. The max-pooling layer reduces a feature map's dimension by partitioning it into same-size neighborhoods (strides) and

generating a smaller feature map replacing each neighborhood of the original feature map with the maximum value of each neighborhood. A flattening layer converts a feature map to a one-dimensional array of features. A dense layer comprises a set of perceptrons that are fully connected to the previous and following layers. Finally, a dropout layer is used to reduce over-fitting by setting each element of an input array to 0 with a fixed probability called *dropout ratio* during the training. We assume that the CNN can contain at most one flatten layer, which divides the CNN in two subsequent sections with different purposes: *feature extraction* and *classification*. The feature extraction section can contain only convolutional and max-pooling layers, whereas the classification section can only contain dense and dropout layers.

E. Reinforcement Learning for CNN Architecture Design

In RL, an *agent* is able to take an *action* that has an impact on an *environment*, then observe the environment's *state*, and finally receive a *reward* from the environment. The sequence of such steps is called an *episode*. The agent's goal is to maximize the reward obtained from a series of episodes.

RC-TL uses RL to select the optimal architecture for a CNN (i.e., the environment) from a finite and fixed space of admissible architectures, which we call *state space*. The state space is a subset of all the possible combinations of the values that the CNN hyper-parameters can assume. Namely, the state-space dimensions are: (i) The number of layers that make up the CNN; (ii) The type of each layer among convolutional, max-pooling, flatten, dense, and dropout; (iii) The number and size of different kernels applied to each convolutional layer; (iv) The stride of each max-pooling layer; (v) The number of perceptrons in each dense layer; (vi) The dropout ratio of each dropout layer. This state-space can become considerably large depending on the values of the mentioned parameters, making it impossible to test the performance of every CNN architecture in the space. RL provides a method to search for the optimal architecture avoiding an exhaustive grid search.

At the beginning of an episode, the agent can take action a from a subset $A(s)$ of the action space, where $A(s)$ depends on the currently observed state s. Every action always adds one layer to the current CNN and fixes the added layer's type and parameter values so that the action leads the environment (i.e., the CNN) into a new admissible state (i.e., architecture). In order to guarantee the arrival state is admissible, the new layer's type is constrained by a set of rules: (i) First and last CNN 's layers must be a convolutional and a dense layer, respectively; (ii) Convolutional and max-pooling layers can be followed only by another convolutional and max-pooling layer, or by a flatten layer; (iii) A flatten layer can be followed only by a dense layer; (iv) Dense layers can be followed only by other dense and dropout layers; (v) Dropout layers can be followed only by dense layers. After the agent has taken action, the corresponding reward is unknown and must be computed by training the resulting CNN with the cluster's representative users' data for a limited amount of epochs (*exploration training*). The model's accuracy is the reward associated with taking that action in that state.

The agent uses the Q-learning algorithm with an ε-greedy strategy to learn the policy for selecting actions and stops searching the state space for better CNN architectures when it has reached either a *target accuracy* or a maximum number of episodes. The reward for taking each action in each state is computed using the Bellman equation. RC-TL sets $\varepsilon = 1$ at the beginning of the exploration phase as the probability of taking a random action. When the agent has taken a number of actions equal to the maximum number of layers allowed by the state space, it starts over from a state with a single layer and linearly decreases ε. The process is repeated until ε reaches a minimum value ε_0 to complete the exploitation phase. In this way, the agent prefers randomly exploring the space during the first phase of the learning to identify promising architectures and gradually changes its policy to select actions that guarantee higher reward to identify higher-performing architectures. Finally, RC-TL selects the CNN architecture with the highest accuracy among all those explored by the RL agent and trains it, allowing a much larger number of epochs compared to the exploration epochs.

F. Transfer Learning between Cluster Members

After building the RL-CNN trajectory predictor associated with the cluster c, RC-TL transfers the pre-trained reference model from the k reference users to the remaining $|c|-k$ users in the cluster c. Let the layers of the pre-trained reference model for cluster c be $L = \{l_1, l_2, \ldots, l_h\}$, where the layers l_1 and l_h represent input and output layers, respectively. The system transfers the learned knowledge of the reference model's neural architecture and weights of the first $h-1$ layers $\omega(l_j), \forall j \in \{1, \ldots, h-1\}$, to the remaining $|c|-k$ users in the cluster c. RC-TL transfers the knowledge of the first $h-1$ layers because the CNN's output layer requires a different number of classes (neurons) for each user. The h-th layer's number of neurons for the j-th user is determined from its mobility data, i.e., set equal to the j-th user's number D_j of different base stations appearing in its trajectory. In this way, the remaining $|c|-k$ users initialize their neural network with the transferred reference model and do not require to be trained from scratch. Algorithm 1 shows the overall RC-TL operation.

IV. Performance Evaluation

A. Experimental Setup

We tested RC-TL 's performance on a large-scale real-world mobility dataset provided by Orange S.A., France. This dataset contains data from nearly 1.3 million mobile users, each of which records the time at which a user connected to and disconnected from one of 131 different cellular base stations deployed in the Paris area between July and September

Algorithm 1: RC-TL Trajectory Predictor

Input: Set of clusters C
Output: Trained CNN Models

```
for each cluster c ∈ C do
    for each user j ∈ c do
        Compute regularity ratio ρ_j ← |T_j|/D_j;
    R_c ← {k users with highest ρ_j};
    RL agent searches CNN architecture using R_c;
    Train CNN using R_c;
    Return prediction accuracy;
    for each user j ∈ c do
        if user j ∉ R_c then
            Load pre-trained RL-CNN;
            user j ← RL-CNN;
            Return prediction accuracy;
    Compute cluster's average prediction accuracy;
Compute overall average prediction accuracy over all clusters;
```

TABLE II
FIXED PARAMETERS

Parameter	Values
Representative users per cluster	10%
Batch size	200
Learning rate decay	0.002
Maximum training duration	200 epochs
Early stopping patience	10 epochs
Early stopping accuracy improvement threshold	0.1
Activation function in dense hidden layer	ReLU
Activation function in dense output layer	SoftMax
Discount factor γ	1
Learning rate α	0.01
Early stopping threshold	80%
Exploration training duration	20 epochs
Exploration training validation	10-fold cross-validation 70% data, 30% test
Exploration training target accuracy	80%
Exploration training max episodes number	500
ε_0	0.01

2019 (63 days). For privacy reasons, the base stations' exact locations and IDs and the user identities are anonymized.

In our experimental setup, we fix some learning parameters for CNN and RL (see Table II) and we define sets of CNN hyper-parameters as the RL search space (see Table III). We selected such parameters and searched space because of their popularity in the literature [18]. We implemented RC-TL using Keras[1], an open-source python library for ML. We trained and tested the suggested predictor on UBELIX[2], a High-Performance Computing Cluster at the University of Bern, Switzerland. UBELIX supports the parallel execution of multiple predictions on a set of machines provided with an AMD EPYC 7742 CPU and 4 GB RAM per process.

[1]https://keras.io/
[2]https://docs.id.unibe.ch/ubelix

TABLE III
CNN HYPER-PARAMETER SEARCH SPACE. EACH ROW CORRESPONDS TO ONE OF ITS DIMENSIONS.

Parameter	Values
Number of layers	$4, 5, \ldots, 20$
Number of convolutional kernels	$48, 64, 128$
Convolutional kernel size	$3, 6, 9$
Max-pooling layer stride	$10, 20, 30$
Number of perceptrons in dense layer	$20, 40, 60, 80, 100, 150$
Dropout ratio	$0.1, 0.3, 0.5, 0.7, 0.9$

First, we evaluate the performance of the proposed RL-CNN against other state-of-the-art ML approaches without clustering, meaning that each neural network is built and trained for a single user. In this way, we focus on studying our proposed RL-CNN regarding the tradeoff between prediction accuracy and network build time. The competing predictors we implemented are: (i) GS-LSTM and RL-LSTM, which are LSTM-based trajectory predictors whose architecture is determined through GS and RL, respectively; (ii) J48 and Random Forest (RF), which are non-neural decision-tree-based models.

Afterward, we evaluate RC-TL 's performance employing three different clustering algorithms (k-means, Ward, and Birch) against the non-clustered RL-CNN approach. In this way, we can evaluate the impact of clustering on prediction accuracy and the computational load required for training and study their tradeoff. Each predictor's performance is evaluated and averaged on 100 random users.

B. Evaluation Metrics

We define a set of metrics to evaluate the performance of RC-TL compared to other mobility predictors, namely: (i) The *accuracy* of the prediction as to the ratio between correctly forecasted next locations and the total number of predictions made; (ii) The *build time* as the sum of the time needed to search the model's architecture and the time needed to train the model for ANN-based predictors (for non-neural predictors, the build time coincides with the training time only); (iii) The *resource consumption* as the ratio between the computation required to build one model for each cluster and one model for each user.

C. Experimental Results

The first experimental results show the impact of the neural network and neural architecture search on prediction accuracy and build time.

We firstly compare all models on an individual, i.e., user-wise mobility prediction in which each algorithm predicts the user trajectory, and the accuracy is given in terms of the fraction of correctly predicted data points. Figure 3 shows that

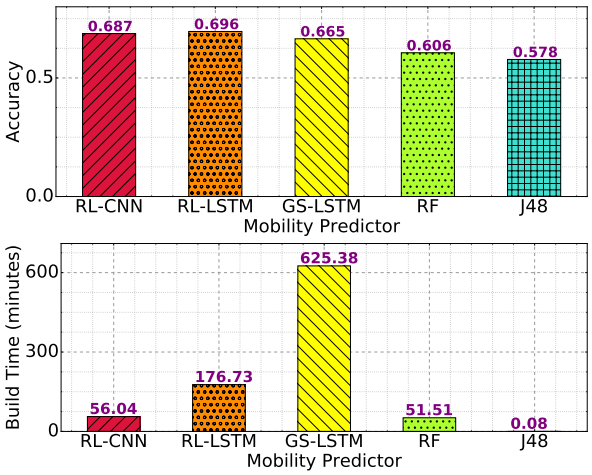

Fig. 3. Accuracy and build time of mobility predictors trained on a single user's data. Results averaged on 100 random users.

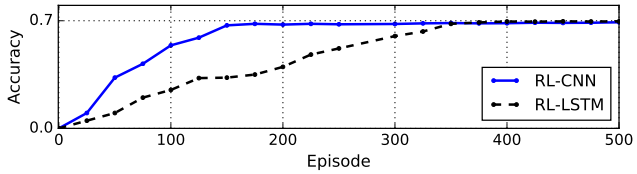

Fig. 4. RL-designed ANNs curves for the average user in the considered dataset. The model accuracy is the agent's reward.

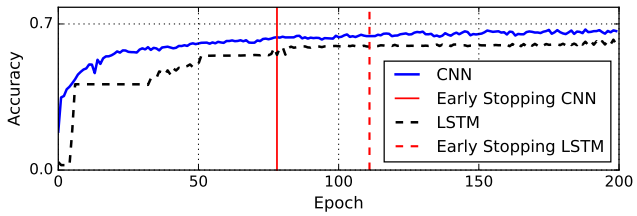

Fig. 5. Learning curves for the best CNN and LSTM models selected by the RL agent for a random user in the clustered scenario.

RL-CNN achieves similar accuracy to RL-LSTM and around 10% higher than RF and J48. Meanwhile, RL-CNN saves 69% of the build time compared to RL-LSTM and is comparable with the RF's build time. Figure 3 also shows that RL reduces the build time by 72% compared to grid-search neural network architecture search, with minimal difference inaccuracy. This behavior can be explained by the relatively short duration of the data collection for the dataset, spanning two months during summer holidays, incurring a more significant degree of exploration in the dataset, as users visit several new places to predict such features challenging. We expect our solution and its accuracy to improve for datasets with a more comprehensive data collection, as it will learn the statistical features of such data points.

We sample the learning curve over the episodes of transfer learning both in the case of applying CNN and LSTM networks in order to compare the convergence of both algorithms. Figure 4 shows that the RL agent can find CNN and LSTM architectures that achieve a 69% accuracy in both cases, even though the RL agent can converge to the best architecture for a CNN in fewer episodes than for an LSTM. This means less computation is necessary by the transfer learning step to find and evaluate NN architecture in order to converge to similar accuracy levels, as shown in Figure 3.

We also sample the individual behavior of the learning process after an architecture has been defined. In Figure 5 we can see the accuracy of the Neural Network (NN) by training epoch in both a CNN and an LSTM network. Figure 5 shows that, between the best CNN and LSTM architectures selected by the RL agent, the CNN learns faster than the LSTM, meaning that it can reach higher accuracy on the test set in fewer epochs. This shows that in this particular case, the CNN architecture can learn the statistical features of the user mobility faster and to a better degree than an equivalently chosen LSTM model. Thus, incurring a lower computational cost to train the architecture and, on a larger scale, better overall scalability of the network. We can conclude that CNNs whose architecture is searched by an RL agent achieve similar prediction accuracy to other state-of-the-art models and can be built in a fraction of the time required by other LSTM-based methods.

The results of the second experiment highlight the impact of clustering on the reduction of computational resource requirements for training. Figure 6 shows that the clustered RC-TL

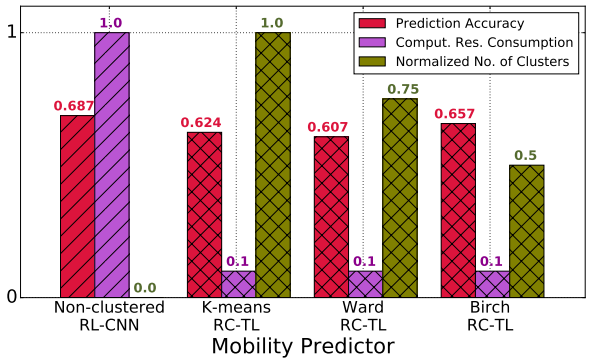

Fig. 6. Performance of the non-clustered RL-CNN predictor, trained on a single user's data, compared with the performance of the clustered RC-TL system using Birch, k-means, and Ward as clustering algorithms, and 10% of the cluster size as the k representative users. Results are averaged over 100 random users.

TABLE IV
IMPACT OF THE NUMBER OF REPRESENTATIVE USERS k ON THE ACCURACY AND COMPUTATIONAL REQUIREMENTS OF RC-TL WITH BIRCH CLUSTERING

k	**Accuracy**	**Computation**
5% of users	60.1%	0.05%
10% of users	65.7%	0.1%
20% of users	65.9%	0.15%

system achieves an almost identical prediction accuracy to the non-clustered RL-CNN predictor, with the Birch algorithm providing the best accuracy among the three tested clustering algorithms. RC-TL trains the model associated with the cluster using the data of 10% of users in the cluster with the highest regularity (representative users). This saves 90% of computational resources compared to training a dedicated model per user. It is worth noting that the Birch clustering algorithm detects half of the clusters detected by the k-means algorithm and 25% fewer clusters than the Ward algorithm, according to the *normalized number of clusters* metric (i.e., the ratio between the number of clusters detected by the considered algorithm and the highest number of clusters detected by all algorithms). The number k of reference users per cluster can be heuristically chosen by testing which values among 5%, 10%, and 20% of the cluster user leads to best accuracy and least computational requirements. Table IV shows the impact of the possible values of k on RC-TL's prediction accuracy and computational requirements. Training a single model per cluster with the data of 10% of users in the cluster and transferring the trained model's parameters to the other 90% of users in the cluster achieves much higher accuracy compared to training on 5% of cluster users and saves around a third of computational requirements compared to training the model on 20% of cluster users.

As indicated earlier, the dataset used in the present work contains mobility information of only two months for over a million users. Due to the limited size of the dataset and the huge variety in sparsity of users' data samples, the achieved accuracy of the suggested predictor is limited by the available dataset's quality. Nonetheless, we proved the superiority of RC-TL in improving accuracy, decreasing training time, and decreasing computational resource consumption with respect to state-of-the-art solutions through case-studying the Orange dataset.

V. Conclusions

Providing a personalized mobility prediction model considerably improves the performance and quality of mobility predictors, but an optimized design of these predictors is a costly task and cannot be feasibly performed for each user in a network. This paper proposes RC-TL, an automated neural network hyper-parameter optimizer. RC-TL leverages the similarities in users' trajectories to build specialized neural networks for entire clusters of users, thus decreasing the resource utilization in terms of CPU time to optimize neural networks for individual users. A Reinforcement Learning agent is used to discover the highest-performance neural architecture for the CNN trajectory predictor within a given search space. Transfer learning is applied to specialize a cluster's neural network for a given user after the best architecture for their cluster is found. We validated the proposed model on Orange's real-world, large-scale mobility dataset. Results show that RL-CNN improves the prediction accuracy by almost 10% on average over the state-of-the-art approaches while its convergence is much faster than other approaches. Moreover, results of clustering-level trajectory prediction through the RC-TL framework illustrate that the system can save up to 90% of computational resources while losing only 3% of the average accuracy.

These preliminary results obtained with the Orange dataset show the potential of the RC-TL model but limit the robustness of the predictor to this specific case study due to the dataset's limited size on users' mobility information. As higher-quality datasets will become available in the future, we will expand the present work to confirm the generalization features of our proposed model. Furthermore, we plan to extend our work in the presence of attention mechanisms, which have attained impressive success in Natural Language Processing and other sequential modeling areas. Finally, we intend to validate the impacts of RC-TL in network and application mobility management, such as handover optimization and service migration.

Acknowledgments

This work was partially funded by the SNF Intelligent Mobility Services project (n. 184690). We thank Orange S.A., France, for providing the dataset used for the experiments.

References

[1] M. Ozturk, M. Gogate, O. Onireti, A. Adeel, A. Hussain, and M. A. Imran, "A novel deep learning driven, low-cost mobility prediction approach for 5g cellular networks: The case of the control/data separation architecture (cdsa)," *Neurocomputing*, vol. 358, pp. 479–489, 2019.

[2] A. Nadembega, A. S. Hafid, and R. Brisebois, "Mobility prediction model-based service migration procedure for follow me cloud to support qos and qoe," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016.

[3] A. Rasouli and J. K. Tsotsos, "Autonomous vehicles that interact with pedestrians: A survey of theory and practice," *IEEE transactions on intelligent transportation systems*, vol. 21, no. 3, pp. 900–918, 2019.

[4] J. K. Lee, Y. S. Jeong, and J. H. Park, "s-itsf: a service based intelligent transportation system framework for smart accident management," *Human-centric Computing and Information Sciences*, vol. 5, no. 1, p. 34, 2015.

[5] W. Zhang, Y. Liu, T. Liu, and C. Yang, "Trajectory prediction with recurrent neural networks for predictive resource allocation," in *2018 14th IEEE International Conference on Signal Processing (ICSP)*, pp. 634–639, IEEE, 2018.

[6] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer, "Generalizable intention prediction of human drivers at intersections," in *2017 IEEE intelligent vehicles symposium (IV)*, pp. 1665–1670, IEEE, 2017.

[7] S. Zamboni, Z. T. Kefato, S. Girdzijauskas, N. Christoffer, and L. D. Col, "Pedestrian trajectory prediction with convolutional neural networks," *arXiv preprint arXiv:2010.05796*, 2020.

[8] A. Shrivastava, J. P. V. Verma, S. Jain, and S. Garg, "A deep learning based approach for trajectory estimation using geographically clustered data," *SN Applied Sciences*, vol. 3, no. 6, pp. 1–17, 2021.

[9] C. Sung, D. Feldman, and D. Rus, "Trajectory clustering for motion prediction," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1547–1552, IEEE, 2012.

[10] N. Deo, A. Rangesh, and M. M. Trivedi, "How would surround vehicles move? A unified framework for maneuver classification and motion prediction," *CoRR*, vol. abs/1801.06523, 2018.

[11] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel, and K. Kuhnert, "When will it change the lane? a probabilistic regression approach for rarely occurring events," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1373–1379, June 2015.

[12] R. S. Tomar, S. Verma, and G. S. Tomar, "Svm based trajectory predictions of lane changing vehicles," in *2011 International Conference on Computational Intelligence and Communication Networks*, pp. 716–721, IEEE, 2011.

[13] N. Nikhil and B. Tran Morris, "Convolutional neural network for trajectory prediction," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.

[14] T. Elsken, J. H. Metzen, F. Hutter, *et al.*, "Neural architecture search: A survey.," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[15] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[19] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-rl: Hyperparameter optimization by reinforcement learning," *arXiv preprint arXiv:1906.11527*, 2019.

[20] Y.-Y. Chen, A.-J. Cheng, and W. H. Hsu, "Travel recommendation by mining people attributes and travel group types from community-contributed photos," *IEEE Transactions on Multimedia*, vol. 15, no. 6, pp. 1283–1295, 2013.

[21] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664–675, 1977.

[22] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang, "A review of moving object trajectory clustering algorithms," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 123–144, 2017.