Reinforced-LSTM Trajectory Prediction-Driven Dynamic Service Migration: A Case Study

Zhongliang Zhao, Negar Emami, Hugo Santos, Lucas Pacheco, Mostafa Karimzadeh, Torsten Braun, *Senior Member, IEEE*, Arnaud Braud, Benoit Radier, and Philippe Tamagnan

***Abstract*—Predicting user behavior is the cornerstone of intelligent services and applications for providing and optimizing services over mobile networks. In modern edge computing scenarios, contents and services will be ordered close to end-users and will be highly sensitive to user mobility. Deep Learning models have had significant success in performing prediction tasks. However, providing reliable predictions for real-world networks in scale requires the Neural Architecture Search to be optimized on a user basis. In this work, we present an LSTM-based mobility predictor to improve the trajectory prediction accuracy. To speed up the model convergence rate, we employ a reinforcement learning technique to automate the selection procedure of the best neural network architecture. To further accelerate the reinforcement learning environmental search procedure, we transfer the architecture knowledge learned from a teacher LSTM to a student LSTM via a transfer learning framework. Furthermore, we showcase the possible improvements of edge-computing enabled networks, in the form of a predictive handover algorithm that applies the prediction information to reduce the handover-failure rate, as well as handover-triggered service migration in edge computing layer of the network. Experiment results prove the efficiency of the proposal, its impacts on improving ping-pong handover, and the service migration.**

***Index Terms*—Service migration, handover optimization, trajectory prediction, recurrent neural network, reinforcement learning, transfer learning.**

I. Introduction

UPCOMING generations of mobile networks, such as Beyond-5G (B5G) and the 6^{th} Generation of Mobile Communications (6G) are already being discussed in industry and academia. While design and implementation details are being decided, it is clear that Artificial Intelligence (AI) plays a significant role in network management and user behavior prediction. Mobility predictors have been used in a wide range of applications, from targeted advertising to network management. Forecasting user behavior is crucial for networked systems to allocate resources and to provide users with a high Quality of Service (QoS) for their applications. While the mobility information of the users may be given in terms of their connection points in the network, we can obtain such information in a finer granularity in next-generation networks, which are characterized by more dense Base Station (BS) deployment and a smaller coverage area for each BS. Thus, user mobility not only can be described by the sequence of small cells the users connect to in their trajectory, but the frequency of connection and disconnection from base stations (i.e., handovers) is greatly increased. However, the denser deployment of BS incurs in the more frequent execution of Handovers (HOs), causing a signaling overhead to the network, as well as frequent disconnections from the contents and services being consumed. In this context, alternative mobility management techniques can be proposed for next-generation networks leveraging the opportunity of learning and making predictions over the users' behavior (*e.g.* the mobile network can predict the user mobility to detect incoming HOs events and allocate and release resources in a more optimal manner). In this context, not only the network access is much more distributed than in previous generations, but the computing power as well. Thus, services and applications must be designed accordingly to such paradigm shift. Based on this knowledge, Intelligent Transportation Systems (IT) enable efficient techniques for smart cities to manage city traffic, accidents and suggest safer, shorter, or less congested routes to citizens or tourists [1], [2].

In this context, modern applications have very strict requirements in terms of storage, computing, latency, and bandwidth. Applications such as Autonomous Driving and Driving Assistance, Extended Reality (XR), Immersive Holographic communications, and others impose the need for the network infrastructure to provide high-quality access to edge servers which will support such applications. In this direction, various heterogeneous services will be offloaded from the cloud to servers located at the edge of the network, which are closer to end-users, constituting another challenge from the mobility management standpoint, constituting the notion of Multi-access Edge

Computing (MEC). Since the edge computing paradigm is geographically distributed, its services are sensitive to users' mobility. As users move to different areas in the city, the services being consumed in an edge server can be disrupted, or be located many hops away from the user, reducing the QoS of applications. Furthermore, the computing power at the edge of the network is still inferior to traditional cloud computing and must be carefully managed. Service migration is one of the most used approaches for keeping critical services close to users even as they move through the scenario, thus improving QoS levels for the applications they consume. User mobility in these edge-enabled scenarios raises the need for services to be migrated to keep QoS requirements for each service under acceptable levels. However, since service migrations typically occur in a reactive manner after HO events, this can cause disconnections and interruptions to user services. An efficient service migration scheme must perform the necessary migration operations in a predictive way to guarantee service continuity for end-users. This requires the usage of an efficient and robust mobility prediction scheme to enable proactive service migrations to a user's future locations.

Service migration highly depends on other mobility management aspects of the network, such as HO executions. In mobile networks, HO is caused primarily by user mobility, and the handover process can be optimized for better QoS for the end-user with a prediction mechanism in place. In this context, the network can offer predictive resource allocation and release, as well as optimize the users' connection points and routes with a predictive view of the users' trajectories. HO decision is a crucial task for maintaining a good QoS and service continuity for end users. When the user moves to different areas, the network has to react to routes and topology changes and adapt it to assure service continuity. The HO procedure demands several signaling protocols between the mobile user, source/target BS, and core networks. Naturally, more frequent HOs might increase delay, decrease throughput, and cause signaling overhead. Instead, a proactive HO mechanism can be designed by integrating mobility prediction and knowledge of the user's next location into the decision. With this design, source and target BS exchange messages before the mobile user switches the connection [3]. Mobility management, and especially the HO procedure impacts the users' experience and the metrics of services being consumed by mobile users. Intelligent mobility management approaches can make use of Artificial Intelligence (AI) top guarantee service continuity with high QoS.

Modern networks face a vast stream of data continuously produced by users and the network itself, such as location data, users' movement patterns, and even the services being consumed by end users. Recently, data analysis, pattern recognition, and prediction tasks are mainly based on Machine Learning (ML), which have improved accuracy due to their advanced algorithms. The dynamic features of mobile user movements, the spatio-temporal dependencies of the urban environment, and time-varying traffic patterns create several challenges in adapting ML algorithms for mobile user location prediction. An ML predictor can learn in terms of the spatio-temporal dependencies and statistical characteristics of the data from mobility traces. Recurrent Neural Networks (RNN) and its variant Long Short Term Memory (LSTM) have shown the outstanding performance in time-series prediction such as Natural Language Processing (NLP) and mobility prediction [4]. However, Deep Learning (DL)-based approaches' performance mostly relies on the design of Neural Networks (NN) architecture and choice of hyper-parameters. Hence, Reinforcement Learning (RL) can be an alternative for human search to find the most suitable architecture [5]. Such approaches are not exclusive and can be used in combination with one another, as presented in this article, including for means of mobile networks optimization.

RL is an unsupervised learning approach that attains the best combination of hyper-parameters by rewarding the high-performance architectures and punishing the low-performance ones. To speed up the convergence rate of the RL-based predictor, we need an accelerating mechanism to optimize the process of finding the best architecture. We found that Transfer Learning (TL) can significantly speed up architecture optimization [6]. TL transfers available knowledge (i.e., architecture hyper-parameters, and weights) learned from a trained predictor to a newly suggested architecture. This helps the new architecture to avoid initializing neural weights from scratch, and, subsequently, the training phase can be done faster. Consequently, a robust mobility predictor developed based on both DL and RL techniques can provide a proactive HO mechanism guaranteeing high QoS and service continuity for network applications such as service migration.

The main contributions of this article are as follows:

- We propose an individual user trajectory predictor unaware of geographical coordinates of anonymous mobile users and BSs to estimate user's future BS connection by applying an LSTM predictor. To automate NN design and hyper-parameter selection, we design an RL-based technique. We also propose a TL scheme to speed up the convergence rate of the proposed Reinforcement Learning based LSTM (RL-LSTM) solution reducing the the training time by up to 3.5 when compared to state-of-the-art algorithms.
- We introduce a service migration framework for edge computing scenarios, which takes advantage of the proposed RL-LSTM solution. Experiments using a real-world dataset attest the efficiency of the scheme in network operation and optimization for delay-sensitive services on a MEC environment.
- We conduct experiments with an anonymized real-world dataset from a mobile network operator. Results validate the performance of our predictor and its impacts on improving HO and service migration applications.

The rest of this work is organized as follows. Section II reviews related works. Section III describes the dataset that is used. Section IV develops a service migration scheme coupled with an intelligent edge-enabled scenario that improves migration performance. Section V describes in detail a recurrent NN-based LSTM trajectory predictor, which is optimized through RL and TL techniques. Section VI examines how the

HO in a mobile network can be improved by mobility prediction. Section VII presents the evaluation methodology and assesses the performance of the mobility predictor and its applications in an edge-enabled network. Section VIII summarizes the contributions of this work.

II. Related Work

A. Real-World Mobility Datasets

Mobility datasets collected by network operators are important information sources to infer users' mobility patterns. The datasets normally include information of millions of users distributed over geographical areas. The basic information usually stores an user's identification, the timestamp of the event, the BS identification related to an event [7], [8]. Custom features added to basic information are provided in terms of event granularity, event duration, type of the application (e.g., call detail records and mobility data), collection period, etc.

Previous real-world Call Detail Records (CD) datasets explore the mobility pattern of individuals logged whenever an user made/received a call or a text message [7], [8]. Gonzalez *et al.* [7] analyze two datasets with more detailed information of users, such as an average service area of 3 km^2 and 30% of BSs covering smaller areas up to 1 km^2. Chen *et al.* [8] provide mobility trajectory reconstruction with GPS coordinates of BSs as well. However, both works rely on users' GPS data, which might pose considerable privacy risks. The storage of user sensitive event data, such as BS identification and the geographical position is not likely to be provided for privacy concerns. The Orange dataset, discussed in more details in Section III, protects user privacy by anonymizing users and BS identifiers, as well as removing GPS data of users and BSs. Therefore, user trajectory prediction becomes more challenging due to the lack of geographical coordinates.

B. Service Migration

The decision-making of service migration on MEC environments brought significant challenges in terms of large-scale experiments and more involvements with AI techniques [9]. Therefore, performing service migration of anonymous mobile users lacking geographical coordinates provides higher accuracy on how edge-enabled scenarios can occur than AI techniques considering more detailed information of users and mobile infrastructure positions. Pure LSTM-based approaches normally take long period of time to adjust layers to improve accuracy, as proposed by Jing *et al.* [10] for accurate prediction of cloud resources for service migration purposes.

Many works focus on service pre-migration in vehicular scenarios, such as Yu *et al.* [11], which propose a migration decision executed offline for services in mobile edge computing. The work uses a mobility prediction scheme to minimize the average latency of the service in the long term. The algorithm, while finding an optimal solution, may have limitations if being used in real-time, as the processing is assumed to be complex. Zhang *et al.* [12] proposes a deep-belief network for prediction of tasks in a cloud computing environment. The authors argue that traditional prediction techniques are far too simplistic for modern networking scenarios. Thus, modern 5 G and edge computing networks require a robust predictor in order to offer satisfactory network performance. Concerning predictive migration for an edge computing-enabled scenario, some works have proposed solutions with certain limitations. Liang *et al.* [13] showcases the opportunities for optimizing resource allocation in a wireless-access edge computing network. The authors propose a joint computing and communication resource management in order to achieve high QoS for end-users. Integer programming is used to provide an optimal solution for the allocation of computation and communication resources, and simulation results achieve near-optimal performance. However, complex optimization procedures are not feasible to be executed in real-time network management, and in real systems are often substituted for lightweight heuristics without significant decrease in QoS. Furthermore, Zhang *et al.* [14] propose a genetic algorithm-based task migration solution for pervasive cloud computing. However, such solution must be re-evaluated for modern B5G edge computing scenarios. Li *et al.* [15] propose a joint service migration and caching for a SDN-enabled edge network. In this context, the proposed mechanism focuses on providing reliable multimedia streaming for users at the edge of the network based on a Q-Learning decision policy which dynamically learns a caching strategy. However, the system does not make usage of predictive mobility information about the users' connection points or geographical position, which could provide the system with proactive management capabilities and increase the perceived QoS for users.

Ouyang *et al.* [16] tackle the problem of keeping services close to users in edge computing scenarios, where user mobility is unpredictable. The system does not need prior information about user mobility statistics, as it applies real-time optimization in order to reduce the complexity of the problem. The solution aims to reduce overall migration costs with specialized mobility models. Also, Gao *et al.* [17] propose a heuristic-based migration algorithm to serve users with varying deadlines, considering user-generated data and the contact patterns between the nodes. Despite employing mobility models in the decision, the proposed solution lacks QoS and radio resources support for applications and services.

C. Handover Optimization

Xu *et al.* [18] propose a delay oriented cross-tier HO skipping scheme, i.e., a handover algorithm that minimizes the number of HOs in order to maximize the performance of low latency applications in ultra-dense networks. Their work derives an analytical expression for users' adequate capacity during the HO execution and proposed a resource allocation scheme in target BSs to reduce blocking probability. It does not employ predictive schemes or mobility information into the decision, which may improve the decision quality and positively impact user Quality of Experience (QoE).

Other works have also applied a prediction-based HO algorithm to mobile scenarios with some success. Gong *et al.* [19]

propose a multi-criteria HO algorithm for heterogeneous networks. Considering multiple criteria in the decision process can be a complex process, and some techniques were developed to balance the parameters based on degrees of importance, such as the one applied by the authors: Analytic Hierarchical Process (AH) attributes weights to each parameter based on predefined degrees of importance. Based on these schemes, the cross-tier HO performance is improved in terms of failure probability and ping-pong rate (*i.e., when a user disconnects and connects multiple times from a BS or a group of BSs in a short period of time*). To make the decision more reliable and avoid unnecessary HOs, the authors also consider previous measurements in the process; however, this makes the proposed solution not as fast-reactive as necessary on challenging scenarios. Mandour *et al.* [20] also provide a prediction-based HO optimization mechanism. The authors choose a scheme based on the reference signal received power, reference signal received quality, and mobility parameters for mobile users. The work aims to eliminate excessive and redundant HOs during a user's path in a small cell network. However, when considering a small cell network, a higher number of HOs is expected to keep service and coverage continuity for the users, thus simplifying mobility prediction and a mechanism to avoid HO execution may not be suitable for all scenarios. In such a case, a more accurate and reliable prediction is essential for a good HO algorithm. In [21], authors propose a group-based pre-handover authentication schemes for 5 G high-speed rail networks.

D. User Mobility Prediction

Mobility prediction is a critical component in intelligent cities and transportation services. Future trajectory prediction of mobile users (either pedestrians or vehicles) can significantly improve urban traffic management, efficient navigation, route recommendation, and safety. Simultaneously, location prediction services are also deployed in many network applications and have enabled proactive mobility management, HO optimization, content migration, and resource management. In general, trajectory prediction can be categorized into two classes: physics-based and maneuver-based models [22]. Physics-based prediction models are mainly based on statistical models such as the family of Bayesian filters and Kalman Filters (KF). The authors in [23] implement KF to predict future trajectories of mobile users. On the other hand, maneuver-based predictions are mainly based on ML algorithms [24]. Heuristic-based classifiers [25], Markov models and Random Forest (RF) classifiers are all examples of maneuver-based trajectory predictions through different ML algorithms. Recently, it was shown that RNN and their variants including Gated Recurrent Units [26] and LSTM [27] are best fitting DL approaches for trajectory prediction from mobility data, which is considered as a form of time series data type. Both of the LSTM and GRU models address the RNNs' vanishing gradient problem. LSTMs have more sophisticated architecture and achieve more accuracy in longer sequence predictions whereas GRUs perform faster operations and consume less memory. Therefore, LSTMs are more preferred for applications that require accurate trajectory prediction. Authors in [28] use LSTM RNNs to estimate the pedestrian future trajectory. Alahi *et al.* [29] present a *social* LSTM that combines physics and maneuver-based estimation algorithms to forecast human motions in crowded spaces.

Despite many efforts on RNN-based predictors, designing an efficient NN architecture remains an open challenge. The above-mentioned DL predictors have designed neural networks heuristically, which does not guarantee the optimal performance. Hyper-parameter optimization is mostly addressed through search methods such as grid search, random search [30], and Bayesian search [31]. However, these techniques are too naive to guarantee the optimal performance. For instance, grid search requires an immense amount of computational power to try all possible combinations of the search space or random search cannot guarantee the optimal neural architecture since the technique randomly eliminates some states of the search space. In this direction, Zoph *et al.* [5] show for the first time how RL can be used as a more robust optimization and search method with respect to above-mentioned naive methods. Zoph *et al.* [5] apply a RNN-based RL to automate the NN's architecture design for the field of image classification.

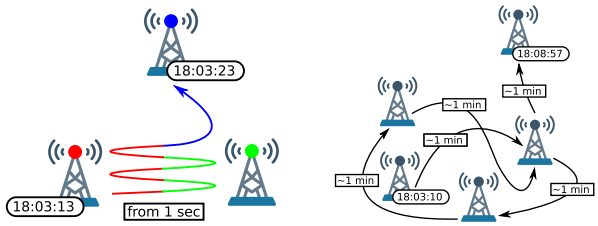
In this paper, we propose a Q-learning-based RL for the task of mobility prediction for the first time. Our proposed mobility predictor could automatically learn the best NN architecture and optimize the NN hyper-parameter selection, which is applicable for real-time applications, where the future location prediction is of essential importance.

III. Orange Dataset

A. Description

This work is based on a real-world dataset provided by the French Orange SA telecommunications company. The Orange dataset is a large-scale dataset that includes nearly 1,300,000 anonymized mobile users' cellular connection information with 131 base stations during 63 days (from July 2019 until September 2019) near the city of Paris. For each user, this dataset holds information regarding user Global eNB id as the anonymized unique ID of a connecting BS, start and end time of the user connection to a BS, and anonymized IMSI as the user ID. Note that we pick the connections' starting time as user time stamps.

The main challenges regarding data preparation are the immense data size and to discriminate characteristics of the dataset. In the following, we summarize some of the Orange dataset's particular characteristics, which makes it challenging to apply NN algorithms. The Orange dataset has a size of 600 GB, including information of more than a million users, which are not homogeneous in terms of the data sampling rate. The collected mobility traces do not include equally-sized granularity of time, which means that sometimes there is only one sample per minute, and there are many samples within another minute. In other words, users frequently change their cellular connections and bounce among surrounding BSs. However, short connection times (few seconds) with BSs might not be caused by user mobility. This behavior could be due to the fact that users connect to more than one of the surrounding BSs



(a) An user performing several ping-pong HOs between surrounding BSs within a minute

(b) Consecutive connections to the same BS and cycle of connections

Fig. 1. User HO and mobility pattern.

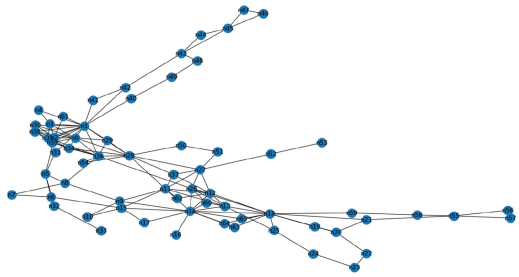


Fig. 2. Re-constructed topology of an user's connected base stations.

or perform ping-pong HOs between neighbor BSs due to varying received signal strength. In this work, we consider a ping-pong HO, according to the definition of Tartarini *et.al.* [32], as a disconnection and re-connection to a BS within 4 seconds. Fig. 1(a) illustrates how users suffer from ping-pong HOs among surrounding BSs within the same minute. Fig. 1(b) shows how sometimes users hop back to the first BS within a short time period, forming a cycle of connections. The reason behind the frequent ping-pong HOs and bounces in the Orange dataset is unknown to us due to the lack of precise locations of users and BSs, signal power, speeds, and direction of users movements. Since the Orange dataset does not provide location and GPS coordinates of BSs, to visualize the BSs' distribution, we extract the relative topology from the mobility traces to re-construct the relative BS topology. From each user's trace data, we detect unique visited BSs and then for each unique BS, we extract all consecutive connections and assume them as the BS's neighbor nodes. Finally, we assign relative coordinates to all the nodes (BSs). Fig. 2 shows an example of a re-constructed topology from the trace data of a random user within 30 days. During this period, the user has moved among 101 different BSs.

B. User Filtering

To implement and test our mobility predictor, we need to pick a group of users, which represents the characteristics of the total dataset, but, at the same time, are containing the minimum amount of required data samples to train a NN. The Orange dataset contains users with heterogeneous data quality. Some users possess a large number of data samples, while others have a small number. We cluster users based on their number of data samples into two classes of good and bad quality users and then randomly select a group of users from the class of good quality users. However, NNs are designed to work with data of different level quality; users with more data samples have more chances to achieve higher prediction accuracy. To identify the good quality users, we set the minimum threshold equivalent to 1600 data samples per user over a total of 63 days. This threshold is selected empirically based on the specific characteristic of the dataset. In this dataset, we frequently observe the appearance of the same BS in consecutive timestamps. In such cases, we keep the first unique sample (BS) and remove redundant ones. Therefore, the threshold on the number of data samples is chosen in such a way that after cleaning duplicated successive BSs, there are still enough data for training the LSTM NN. After applying the threshold on the number of user samples, we lose almost 86.2% of users, meaning that only 13.8% of users contain enough data to be trained and achieve acceptable accuracy.

The achieved percentage of 13.8% corresponds to 180,000 good quality users out of the total 1.3 millions users. This percentage is considered as our sample space for training and testing the mobility predictor. After filtering the good quality users, the Orange dataset keeps a notable proportion of users compared to two well-known studies, which kept only 0.45% and 1.67% of the total data respectively [7].

C. User Selection

In the good quality subset, we choose a group of 100 random users. Within the chosen group, different users have different mobility patterns. Some users travel more often per day, on average, and switch between BSs more frequently while others move less or are quite stationary. Some users show periodic behavior in their trace history while others have irregular patterns. Some users contain data for all days during the total 63 days of the dataset, while others miss data during some days. This article demonstrates how the proposed mobility predictor applies for any users with high or low quality, periodicity, and mobility patterns and achieves reasonably high accuracy.

Furthermore, to better infer users' mobility characteristics and extract the most effective features on prediction accuracy, we evaluated the total number of BSs, the number of days that a user contains data, and the average and standard deviation of daily visited unique BSs. From the analysis we understood that the average number of daily visited BSs is the most relevant attribute of each user's achieved prediction accuracy. We observed that selected 100 random users visited a different number of BSs per day, ranging from 1 to 24. For each user we defined a correlation ratio between the achieved prediction accuracy and the mean number of daily visited unique BSs. Afterwards, we cluster users as four groups of very-high-mobility, high-mobility, medium-mobility, and low-mobility users by defining four empirical thresholds that evenly divide the spectrum of correlation ratios.

IV. RL-LSTM-Based Service Migration Framework

A. Overview

This section discusses the service migration strategy, so called Reinforcement Learning based Service Migration

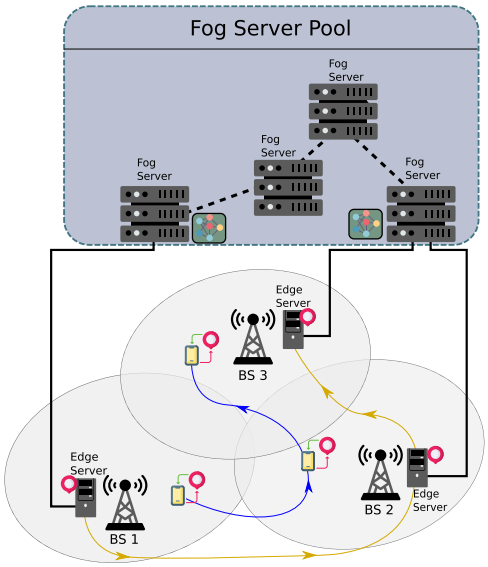


Fig. 3. Multi MEC RLSM Scenario.

(RLSM), driven by the user mobility prediction. RLSM relies on individual user mobility prediction to perform proactive service migrations, and guarantees service continuity. Furthermore, we consider the likelihood that User Equipment (UE) will connect to a certain BS in the future to infer the best edge server to serve the user. We consider a framework with two steps: monitoring and assignment.

We assume an edge-enabled network such as shown in Fig. 3. We see that users access the network through BSs with different coverage areas and follow some mobility pattern. Such BSs are directly connected to edge servers that are capable of providing low-latency computing to the users affiliated. However, edge servers are limited in computing power and cannot support many users at the same time. Thus, edge servers are placed a level above in the hierarchy and can be accessed with decreased latency compared to traditional cloud data-centers. We consider that computing nodes can be accessed close to the end-users, in what we call the Fog Server Pool. We assume that prediction models for users are pre-trained in the Fog Server Pool and are accessible to all other servers in the network after training.

B. Framework Operation

RLSM considers an edge-enabled network scenario, where users can consume services running on edge servers. As shown in Fig. 3, users move and perform HO operations, the topology of the network may change, and the routes from user to service may be sub-optimal. These services are orchestrated and allocated to an appropriate server. Therefore, we consider an orchestrator entity with knowledge of the network deployment, servers' QoS and resources, as well as users' historical and future mobility information. In this work, we assume a service-agnostic approach in which services are encapsulated in VMs or containers, according to the EdgeIoT paradigm [33]. In such scenarios, the conditions of the back-haul and access networks are highly dynamic. User mobility and BS connection fluctuations with edge servers and network resources may induce errors and decrease QoS for end-users. For this reason, the network must monitor user QoS levels and perform the necessary operations, such as migration. This requires continuous re-evaluation of the best edge server to ensure service continuity and perform the necessary migration operations.

Service migrations can be triggered by user mobility or when the user's current server is no longer capable of maintaining appropriate QoS levels for them. In such cases, a migration procedure is started on the server by a controller based on the user's future location and the server's QoS performance. To define the servers' QoS performance, we must consider the type of application being considered in terms of QoS requirements. The server lists the delivered QoS statistics it achieved for applications with the same requirements and feeds these values into an Exponential Moving Average component, which attributes more weight to recent measurements to reflect the recent network conditions according to (1), where Q_t is the QoS average at a time t, and Q_s is defined as the QoS score achieved by the server s at the time of the measurement. The number of the measurement is given by variable n, and $\alpha \in [0, 1]$ is a decay factor. We define the QoS parameter Q_s for the server as whether it is able to provide the applications it is serving with the necessary requirements. Server QoS Q_s is defined as the fraction of applications running in the server provided with their minimum requirements.

$$Q_t = \begin{cases} Q_s, & \text{if} \quad n = 1 \\ \alpha Q_s + (1-\alpha) Q_{t-1}, & \text{if} \quad n > 1 \end{cases} \tag{1}$$

The algorithm functions as follows: considering an UE u in the set of UEs U, u may be associated with a mobility prediction model that can be applied to find when a HO will be triggered for a given UE. This is achieved in conjunction with the HO algorithm operating in the network, which must report imminent HOs to the service migration framework. Each time the UE u moves, there is the possibility that a HO will be triggered. Thus, the orchestrator predicts the user's next HOs. The prediction is a forward propagation task. Therefore, it is not so computationally expensive and may be executed regularly. The prediction outputs the user's next BS. Given this information, RLSM performs a lookup to find the edge servers associated with u's next BS.

RLSM Monitoring: The first decision of RLSM is whether a migration is necessary or not. Migration may be necessary because of mobility, as the service becomes more distant from the server, and the latency increases or the servers can no longer support the application QoS requirements. The monitoring collects the user's predicted position in a given time window and checks whether a user is likely to connect to another BS. If the current server can not meet QoS requirements, a migration process is triggered. Algorithm 1 presents the RLSM monitoring process.

RLSM Assignment: The essential characteristic of the assignment is whether the target server can provide the latency and computation requirements and if so, the migration can be made promptly. RLSM assumes that each edge server can assess the

Algorithm 1: RLSM Monitor.

input: Mobility prediction model for each user in the network.
output: Migration decision for each user.

```
while user is connected do
    Perform mobility prediction;
    Estimate when HOs will be triggered based on the predicted
     user trajectory;
    if HO is eminent then
        Perform migration decision;
    Measure QoS;
    if QoS is below the threshold then
        Perform migration decision;
```

Algorithm 2: RLSM Assignment.

input: User not provided with minimum requirements, list of available servers.
output: ID of the best server for the user, migration operations.
Data: Minimum requirements of the service

```
List available servers;
Remove servers lacking resources for the UE application from
 list;
for Each available edge server do
    Get QoS for the server;
while Server has not been chosen do
    Get the closest server to the UE's future location;
    Estimate migration time;
    if Migration can be done before the UE's arrival and Server
     QoS is above threshold then
        Choose this server as target;
    else if Current connected server QoS is below threshold and
     Server QoS is above threshold then
        Choose this server as target;
    else
        Remove this server from list;
Perform migration;
```

bandwidth of the link to every other edge server and uses this available bandwidth between the edge servers to estimate the time it would take to migrate the UE session to candidate edge servers. The bandwidth available between two servers is probed periodically, and the values are used to estimate the time to migrate a service between such servers. RLSM has relatively low complexity. The algorithm's complexity is proportional to the product of the number of UEs and the number of edge servers. As soon as RLSM detects that a migration is necessary, the algorithm must evaluate all available servers in the user's future location about the server's resources and the time to migrate the service to that specific server. Algorithm 2 shows RLSM Assignment of an edge server to which an UE session shall be migrated.

The protocol for the execution of the migration algorithm is described in more detail in terms of the sequence diagram shown in Fig. 4. We can see the mobility predictor as an entity that receives the current connected BS for a given user and reports to the user and their current BS. After the handover and a migration decision have been executed, the source edge server and the

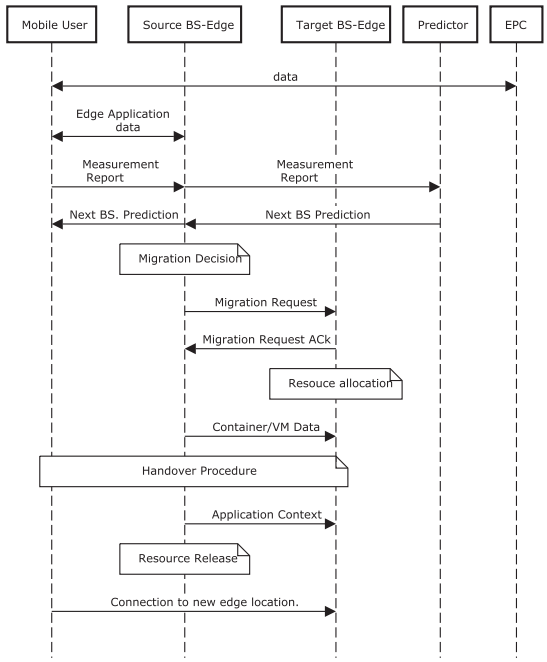


Fig. 4. Sequence Diagram for Migration Procedure.

target edge server must negotiate the migration procedure in terms of resource allocation, the transfer of the VM or container with the service and session information. After such transfer has been completed, the memory pages and application context that have changed since the start of the process are sent to the target edge server. In the final stage, the resources in the source server can be released and a ready to serve another UE, as the migrated user connects to the new server and the process is completed.

V. Trajectory Prediction Through Self-Learning LSTM

A. Problem Statement

In this section, we present a DL-based trajectory predictor based on the combination of LSTM, RL, and DL algorithms. The NN design procedure is quite complex despite the high accuracy and robustness of RNN (e.g., LSTM) predictors for time series data. The architecture of a NN refers to the number of hidden layers, the number of neurons per layer, the way they are connected, and the other DL design hyper-parameters. As expected, the task of hyper-parameters selection requires a significant amount of human effort and time. Previous works have determined the NN architecture in a heuristic way, which does not guarantee optimal performance. Hence, to address this issue, we take advantage of RL and TL approaches to automate and speed up the NN architecture design procedure. In this work, we refer to the Neural Architecture Search (NAS) framework [5], where the RL-based controller suggests various architectures to the predictor. Afterwards, the algorithm's output is fed back as a reward and is used to update the controller to generate better architectures over time. Our proposed Transfer Learning and Reinforcement Learning based LSTM (TL-RL-LSTM) mobility predictor is described by the following three steps:

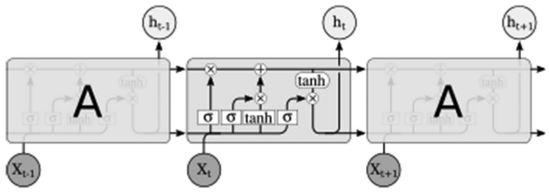


Fig. 5. Stacked LSTM network architecture.

i) Student LSTM predictor aims to fulfill high accuracy for the mobility prediction task.
ii) Q-learning controller as a RL agent proposes better architectures for the student LSTM to maximize the overall expected prediction results.
iii) Knowledge transfer from a previously trained LSTM model (teacher LSTM) to a new architecture (student LSTM) accelerates the architecture search process.

B. LSTM Predictor

Mobility trace data are considered as time-series data that have been collected at different points in time. Time series prediction can be generalized as a process that extracts useful information from historical records and estimates future values. Within the DL framework, RNNs have been extensively applied for time-series prediction [4]. However, conventional RNN suffers from a lack of long-term memory to capture and remember long-term dependencies of time series data. Thus, optimal prediction accuracy is not guaranteed by them. In this work, a specific kind of RNN predictor, so-called LSTM, has been proposed to solve the problem of long time-dependencies. An LSTM network is composed of multiple copies of basic memory blocks, and each memory block contains a memory cell and three gates (input gate, output gate, and forget gate). Fig. 5 shows an overview of a LSTM NN architecture through a chain of memory cells known as stacked LSTM. The memory cell is the critical component of long-term memory and is responsible for transferring the information at different time steps. Each of the three gates contains a sigmoid layer in order to decide how much information passes through memory blocks. The input gate controls which part of the input will be utilized to update the cell state. The forget gate controls which part of the old cell state will be thrown away. The output gate determines which part of the new cell state will be output.

In our predictor design, we consider multiple layers of basic LSTM cell units to form a stacked LSTM network. The intuition is that the deep LSTM network can better understand the temporal dependencies of the users' locations. The LSTM unit of each layer extracts a fixed number of features passed to the next layer. An NN with enormous depth (e.g., a large number of neural hidden layers and neurons of each layer) improves the accuracy of the prediction task, which is done by the last fully connected layer. Hence, hyper-parameters have a significant impact on the performance of ML-based prediction algorithms. The optimal choice of the number of hidden layers, neurons per layer, and other NN hyper-parameters requires expert knowledge and takes a long time to be explored. Usually, the hyper-parameters of the LSTM-based mobility predictor e.g., number of hidden layers including LSTM, dense, and droput layers, number of neurons per each layer, and order of layers, have been determined in empirical and heuristic ways, which is not a optimal solution. To improve this, we propose a RL-based approach to automate the process of NN architecture selection.

C. Automation of LSTM Architecture Design Through RL

It is required to fine-tune the NN architecture and choose the best possible combination of hyper-parameters to have a highly accurate LSTM predictor. We implement an RL agent on top of our LSTM predictor to automatically optimize the NN design process without human intervention. RL learns what to do and how to map states to actions to maximize the reward signal. The RL agent is not told which actions to take, but instead, it must find out which actions would harvest the most reward by exploring and exploiting various possible scenarios. The learning process can be modeled as closed-loop as follows:

i) The agent receives state s_t from environments at time t.
ii) Being on state s_t, the agent decides to take a specific action based on the predefined policy and reference table.
iii) With action a_t, the agent updates to a new state s_{t+1}.
iv) Environments return a reward r_t to the agent.
v) The reference table gets updated by the reward corresponding to the chosen action.
vi) The agent learns to take the best actions over time based on the accumulation of reward during the explore-exploit process and updated values of the reference table.

In our research, RL automatically adapts the NN hyper-parameters to optimize the performance of the mobility prediction. If RL is applied to NN architecture design search, an NN with a particular architecture is trained to achieve satisfying accuracy. At the end of each episode or iteration, the network (e.g., accuracy, precision, and recall) is a reward signal for the RL controller. An episode corresponds to one complete sequence of taking an action, updating the state, and receiving the reward. Utilizing the reward for each suggested architecture, the controller would generate improved architectures over time. In this work, we create a Q-learning-based RL controller that seeks to learn the policy to maximize the total reward in order to find the high-performance LSTM architecture. To learn the best policy, the learning agent needs to explore the search space, including *Action space* and *Parameter space*, and afterwards learns the best state corresponding to the best action. However, searching through the immense search space is extremely time-consuming, and due to the limited computational resources, a mechanism to speed up the convergence process is demanded. Therefore, we limit the search space to a finite but still large space of possible architectures to be searched by the RL controller.

On action space, we define some restrictions on learning agents from taking certain actions. For instance, we allow the RL controller to terminate the predefined total number of iterations sooner if the suggested LSTM has already reached a satisfying prediction accuracy (e.g., 80%). Otherwise, the

process will terminate when the learning agent has explored the whole defined search space. Besides, we force the learning agent to have a dropout layer [34] after each hidden layer to prevent over-fitting as a serious issue in ML algorithms. On parameter space, we define a range of possible hyper-parameters (e.g., number of hidden layers, number of neurons in each hidden layer, and dropout ratio) that the learning agent is allowed to try. We define the number of hidden layers as an integer value in an interval of $(0, 5)$. The number of neurons in each hidden layer might be chosen from discrete list of $\{20, 40, 60, 80, 100, 150\}$. Candidate values for dropout ratio fall down in the set of $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. Besides, we use Rectified Linear Unit (ReLU) as non-linearity functions [35] of each NN dense layer.

After illustrating an appropriate search space, the learning agent trains through random exploration, slowly converges, and selects higher-performance architectures. To summarize the learning process, on each iteration, from the predefined set of iterations $t \in \{0, 1, 2, \dots 150\}$, the Q-learning agent in state $s \in S$ takes an action $a \subseteq A(s)$ and moves toward next state $s' \in S$. In this loop, the reward ($r_t \in \mathbb{R}$) corresponding to the action-state is generated, and the Q-table (reference table) is updated. In subsequent iterations, the agent starts learning and suggests better architectures over time. In the beginning, the Q-table is randomly initialized. However, after each episode, it gets updated from the reward signal corresponding to the taken action and the state. The state-action values are denoted as Q-values or $Q(s, a)$ within the Q-table.

In mobility predictions, taking an action means suggesting an NN architecture to predict the future trajectory, and passing to a new state means estimating the performance of the predictor corresponding to the proposed architecture. At the end of each iteration, the reward signal defined as trajectory prediction accuracy feeds back to the Q-learning agent whose ultimate goal is to maximize the total cumulative rewards. In RL, the maximization of the total expected reward is often defined as a recursive problem within Bellman's Equation. In this article, we employ the ϵ-greedy strategy as our explore-exploit policy [36]. In the exploration phase, the learning agent randomly suggests a new architecture to the LSTM predictor. Afterwards, the agent notices high-performance architectures from high rewarded values of the Q-table and begins converging, which refers to the exploitation phase. The controller on each iteration explores (suggests a random LSTM architecture) with probability ϵ and exploits with probability $1 - \epsilon$, so that $0 \leq \epsilon \leq 1$.

The Q-learning agent's design process with epsilon-greedy policy includes choosing the following parameters: the probability of epsilon ϵ, Q-learning rate α, and discount factor γ corresponding to Bellman's Equation. In the first episodes, we set $\epsilon = 1.0$ to guarantee the exploration phase for an agent, and then gradually, we reduce ϵ to 0.01 to move towards the exploitation phase. Q-learning rate denoted as $\alpha \in (0, 1]$ determines the weight given to new information over old information, and discount factor denoted as $\gamma \in (0, 1]$ determines the importance given to immediate rewards over future rewards.

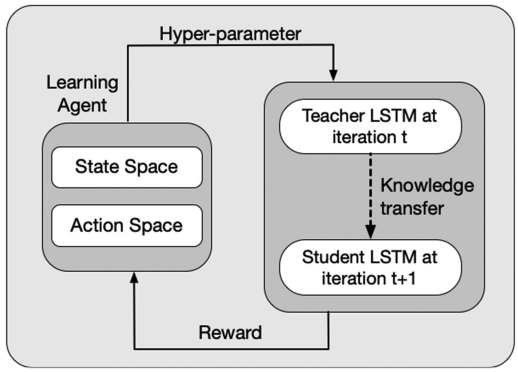


Fig. 6. RL-LSTM predictor system design with knowledge transfer.

D. Expedition of RL-LSTM Through TL

The RL agent suggests possible architectures from the search space to the LSTM predictor to be explored. Although the way RL explores the search space is remarkably faster than grid searching, yet it is very time-consuming. Therefore, in RL scheme, it is not efficient to train each proposed NN architecture from scratch. Therefore, in this section, we present a TL framework, which offers a way to transfer knowledge from the previously trained LSTM predictor (denoted as *teacher LSTM*) to a newly suggested LSTM predictor (denoted as *student LSTM*) in order to speed up the searching process.

In this way, at each iteration, the newly suggested student LSTM can pass the learning phase faster. We apply TL in case of similarities in architectures of teacher and student LSTMs in terms of hidden layers, neurons per layer, and connectivity. This means transferring the knowledge of similar layers from teacher LSTM at iteration $t - 1$ to student LSTM at iteration t. The knowledge here refers to the weights of the teacher LSTM NN architecture that is saved and transferred to the student LSTM to be initialized with. In this article, we employ an adaptation of Net2Net research [37], where authors attempt to transfer the pre-trained predictor's knowledge at iteration $t - 1$ to the new predictor at iteration t.

The teacher LSTM has n layers denoted as: $L = \{l_1, l_2, \dots, l_n\}$, where the layers l_1 and l_n express the input and output layers of the NN. We assume that the RL controller proposes a new architecture to student LSTM, which is exactly the same as the previous one, but contains an extra new layer l'_i. This layer would be implemented between layers with index $n - 1$ and index n (output layer) as follows: $\tilde{L} = \{\tilde{l}_1, \tilde{l}_2, \dots, \tilde{l}_{n-1}, l'_i, \tilde{l}_n\}$. We define the function $\upsilon(l_j) \in \mathbb{N} > 0$ that represents the number of hidden neurons of each layer l_j, so that $1 \leq j \leq n$. Further, we define a weight function $\omega(l_j) \in \mathbb{R}^{n \times m}$, where $n, m \in \mathbb{N} > 0$, in order to build the weight matrix corresponding to NN. We assume that the teacher and student LSTMs have the same number of neurons in each of the layers if: $L \bigcap \tilde{L} := \{l_i \,|\, \upsilon(l_i) = \upsilon(\tilde{l}_i)\}$ for $i = 1, \dots, n - 1$. Thus, we can transfer knowledge from l_i to $\tilde{l}_i$ for $i = 1, \dots, n - 1$. Transferring knowledge means carrying and copying the first $n - 1$ layers' neurons weights from the teacher to the student LSTM as: $\omega(\tilde{l}_i) := \omega(l_i), \forall i = 1, \dots, n - 1$. Fig. 6 shows a high-level system design of the proposed LSTM mobility predictor.

VI. RL-LSTM-Based Handover Optimization

A. Overview

This section details a HO algorithm called Reinforcement Learning-based Handover for Edge Computing (RL-HEC), a service-aware HO algorithm based on a DL mobility prediction. RL-HEC takes advantage of the individual prediction models trained for users to avoid ping-pong HOs, and maximize service continuity. The proposed scheme considers the services being consumed by end-users and connection information to avoid link failures and service disruptions caused by the HO. This is achieved by assuming knowledge of the service instances' locations being consumed by users, since HO executions may require rerouting of such services, compromising service requirements. The proposed scheme performs better than state-of-the-art algorithms.

RL-HEC takes as inputs mobility prediction models of each user and the location of the services in terms of where the session is located in the network topology. We define a ping-pong HO as disconnection and reconnection to a certain BS within an interval of 4 seconds [32]. The prediction model for a given user can forecast with significant accuracy the next connection of a user, such as described in Section V, and thus can be used to predict the occurrences of ping-pong HO and also service migration patterns. We divide the HO procedure into three phases: *(i)* measurement, *(ii)* decision, and *(iii)* execution. In traditional signal-based HOs, the measurement phase comprises the user devices reporting the signal from all the neighboring BSs they can detect. However, this a purely reactive approach i.e. when the user moves from one coverage area to another, they switch BSs accordingly.

At the borders of BSs' coverage areas, signal fluctuations tend to trigger network events such as the LTE A2, A4, and A3 events [38], which are used in traditional HO decisions. This causes unnecessary HOs, such as ping-pong HOs, which are characterized by consecutive disconnections and re-connections within a group of two or more BSs. RL-HEC takes advantage of the future connections predicted by the individual LSTM models to employ a reliable ping-pong avoidance mechanism capable of reducing ping-pong HOs by as much to almost zero, as shown in Section VII-D.

B. Scenario Description

We consider a mobile network, such as the one represented by the dataset used in our experiments, described in more detail in Section III-A. In it, we assume the presence of N UE's, each connected to one of the C BSs present in the scenario. Users move through the scenario and trigger HO events according to the HO algorithm. A given UE $n \in N$ may or may not have an LSTM model m associated with it, depending on the quality of the data available for the UE, as described in Section III-A, and each model $m \in M$ has an accuracy level known by the network. Overseeing the HO process, we consider a HO Manager similar to the 4 G Mobility Manager Entity (MME) or the 5 G Access and Mobility Manager Function (AMF). This HO Manager has access to the individual prediction models for users and their connection history for ping-pong detection.

C. Measurement Phase

The measurement phase of the algorithm is responsible for receiving the necessary inputs for the execution. In the case of RL-HEC, these inputs are the mobility prediction model for the respective user being considered, the location of the services being consumed by the user, and signal measurements. RL-HEC must then assess the quality of the prediction model received, as a low accuracy model can impact the overall performance of the network. We assume that trained models are located within a centralized entity and can be accessed by BSs. Here we must define a threshold T_h as model accuracy, below which the model is not used in the algorithm. If the accuracy is above the threshold, the algorithm uses the trained model for the user. For practical purposes, we use the value of 0.8 as the threshold, as it means that in the vast majority of cases the prediction will be valid, and in the few cases, where it might be wrong, the other parameters of the decision can offer more reliability for the HO decision.

D. Decision Phase

The decision phase in RL-HEC happens within the HO Manager, which is located in the user's serving BS. The HO Manager is a distributed entity running in every BS of the network. Therefore, each BS performs the HO decisions for its users. The algorithm's decision phase can occur independently for each user, as it is a distributed process. Let us consider the case for a single user with a mobility prediction model associated with it. The HO manager assesses the next HO predicted for the given user. In order to avoid ping-pong HO executions, the HO manager checks if the HO in question is a ping-pong HO. After a series of ping-pong HOs, mobile users will usually have a more stable connection in a given BS's coverage area, thus, we must detect in which BS the user will remain connected. Then, before the HO is actually executed, the HO algorithm must notify the RLSM migration algorithm about the upcoming HO, so that any necessary service migrations can be performed in advance. This is possible, because HOs usually occur within overlapping coverage areas of two or more BSs. Thus, the UE's previous BS may still be available for a short period of time.

E. Execution Phase

We consider that a sudden HO execution can be a disruptive event for user applications, as the changes in routes can increase the end-to-end latency between the service's location and the end-users. For this reason, in the execution of RL-HEC, the algorithm waits for any pending service migrations to be finished, thus, improving service continuity. However, there are cases in which a disconnection occurs before pending migration is finished. In such case, the HO must be executed as soon as possible, and a discontinuity for the service being consumed may be inevitable.

Algorithm 3: RL-HEC Algorithm.

```
Data: Prediction depth
for each connected UE do
  if UE has prediction model then
    if Prediction accuracy is above threshold then
      Perform mobility prediction;
      if Handover is expected then
        if Next HO is a ping-pong HO then
          Define best BS as BS in which the client will stay the
           longest;
        else
          Define the best BS as the predicted BS
        Notify migration algorithm;
        while HO not executed do
          Wait for service migration to finish;
          if Disconnection is imminent then
            Perform HO to best BS;
            Return HO status;
          Perform HO to best BS;
          Return HO status;
```

For users with a poor prediction model or without a prediction model at all, a normal signal-based HO decision is made since these users cannot benefit from mobility prediction. However, the performance of the ping-pong avoidance system depends heavily on the accuracy of the models. The model accuracy on the evaluation dataset is discussed in Section VII-C. For now, we assume that for each model, the accuracy at the time of the prediction is known and then compared with the threshold set. The complete function of the algorithm is given in detail by Algorithm 3. We consider that not all user models have the minimum accuracy in performing a good HO decision based on the predictions, so we define the existence of a threshold accuracy for each UE. In the algorithm, the case in which a user has a sufficiently accurate model is shown. The ping-pong HO avoidance mechanism can be accomplished by extrapolating the mobility prediction to the desired depth and checking if the predicted HOs are ping-pong HOs. After a HO has been predicted, the best BS is selected as the user's "stable connection", the one in which the UE converges after the ping-pong HOs. After that, a handover is scheduled.

VII. Experiments and Evaluation

In this section, we discuss the setup details of TL-RL-LSTM. We also examine the performance of our predictor using the Orange Dataset. Furthermore, we evaluate the impact of location awareness on service migration performance in terms of QoS and QoE metrics and the number of ping pong HOs and throughput.

A. Trajectory Predictor Design Details

The design and the evaluation of the suggested TL-RL-LSTM are inspired from the unique characteristics of Orange dataset. The gathered Orange dataset does not include GPS coordinates nor acceleration information of users' movements due to privacy issues. Therefore, the inputs of the designed LSTM trajectory predictor are the sequences of BS IDs and time stamps of the connections. The trajectories can be defined from the data traces based on time or space. Trajectories can include the whole or a fraction of trace duration depending on data collection for each user.

As we explained in Section III, Orange dataset includes mobility data of only 2 months. Therefore, the achieved accuracy of the suggested predictor is limited to the dataset quality. In this work, due to the short duration of the dataset, we have considered each user's whole trace of a few months within all days of the week as his/her input trajectory.

In RL's exploration phase, where the system suggests different NN architectures and searches the most accurate one, we train and test each user's data based on a 10-fold cross-validation approach for few epochs. Splitting data to K folds and then applying cross-validation guarantees that all observations from the dataset find the chance to be trained and tested. Cross-validation is the best solution for datasets with limited data and strengthening the network against overfitting. Afterwards, on each fold, we train 70% of the user's trace data and use the rest 30% to validate the performance of the suggested LSTM. After the RL exploitation phase, when the best architecture has converged, we train the discovered LSTM for 200 epochs to evaluate the performance of the chosen architecture.

Furthermore, we apply the Early Stopping approach to speed up the training process. Early Stopping terminates the training if accuracy optimization gets stabilized sooner than defined epochs. Other hyper-parameters of the TL-RL-LSTM predictor are set as follows. The batch sizes are set to 200, and the predictor's initial learning rate is set to 0.002. To schedule the portion of immediate reward concerning the distant future reward of RL, the Q-learning rate (α) and discount factor (γ) are set to 0.01 and 1, respectively. In this work, since the goal is optimizing the neural architecture, the Q-learning rate and discount factor values are not treated as hyper-parameters.

In this research, the predictors are trained and evaluated on a High-Performance Computing Cluster at the University of Bern in Switzerland (HPC Cluster - UBELIX [1]) on an eight-core machine of Intel(R) Xeon (R) E5-2630 v2 @ 2.60 GHz with 4 GB RAM process environment. The building model uses a single core for each user enabling parallel execution of the predictor for multiple users.

B. Evaluation Metrics

We use accuracy and ML building model time, in other words, the time it takes for the search to converge to an architecture for a given user, as evaluation metrics to examine the proposed trajectory predictor's performance. Accuracy is defined as the ratio between predictor's correct predictions to its overall number of predictions to measure predictor's performance. The building model time in NN predictors refers to the neural architecture search time plus the training time, while, in non-NN predictors, it refers only to the training time. The ML building model time is an important measure for the scalability of a predictor.

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

TABLE I
LSTM ARCHITECTURES SUGGESTED BY RL AGENT FOR HETEROGENEOUS AND HOMOGENEOUS USERS

user ID	total visited BSs /63 days data	neuron/ LSTM layer	dense layers	neuron / dense layer
59	110	16	3 layers	150-40-110
16	105	16	3 layers	80-150-102
10	49	16	2 layers	150-49
81	22	16	2 layers	40-20

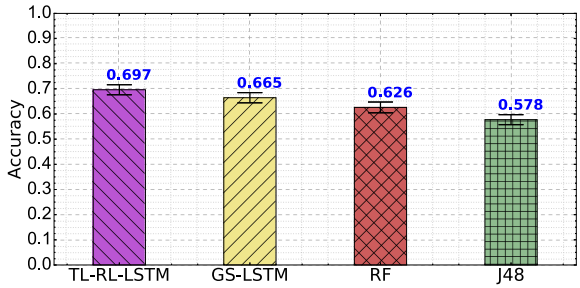


Fig. 7. Average prediction accuracy of 100 users.

C. Trajectory Predictor Experimental Results and Performance Comparison

In this section, we present the experimental results and evaluate the performance of the proposed TL-RL-LSTM predictor. The implemented RL controller searches for the best LSTM architecture for each single user. According to our experiments, the complexity of each suggested architecture is affected by the type of user movement. So, users with large numbers of visited BSs (heterogeneous movement) and less periodic behavioral history have been suggested architectures with deeper hidden layers. In contrast, users with a few numbers of visited BSs (homogeneous movement) and predictable movements got simpler architectures. Table I exposes RL's suggested LSTM NN architectures of 4 different users, from the chosen 100 random users, with different mobility patterns including 2 random heterogeneous and 2 random homogeneous users, respectively.

To illustrate the advantages of our proposed predictor, we compare the TL-RL-LSTM mobility predictor against state-of-the-art prediction techniques, namely: J48 predictor with non-parametric supervised learning method for regression with decision trees, regressive RF predictor that has a random subset of features from the training data points to create multiple decision trees, and Grid Search optimized LSTM (GS-LSTM) neural network. J48 and RF are non-NN predictors and thus, do not require neural architecture search schemes. TL-RL-LSTM and GS-LSTM are two automated NN-based predictors that search for the best neural architecture before training the individual dataset. Unlike RL, which tries to find the best architecture while minimizing the search space, grid search is a computationally expensive and slow algorithm since, it fully trains all possible architecture combinations and then selects the best available choice. However, RL is by nature an optimised architecture search method with respect to naive search approaches, yet it requires a remarkable training time to discover the best neural architecture. Thus, we implemented a TL approach on top of the RL algorithm to further accelerate and optimise the RL-LSTM's searching and training process.

Fig. 7 shows the average accuracy achieved by each of the compared algorithms on the tested dataset. We can see that TL-RL-LSTM achieved 69.7% accuracy, 3.2% better than GS-LSTM in average. Further, TL-RL-LSTM also achieved 7.1% and 11.9% superior performance than the non-NN methods: RF and J48. This is due to the fact that deep learning-based methods can better capture the complex spatio-temporal dependencies.

Fig. 8 compares the performance of our suggested RL-LSTM predictor with and without TL. It can be observed that transferring the knowledge of a pre-trained LSTM layer (teacher-LSTM) to a newly-suggested LSTM layer (student-LSTM) helps the RL agent to converge sooner. So that, TL-RL-LSTM is stabilized, on average, at about the 75th episode (out of 100 episodes), while RL-LSTM starts to stabilize around 90th episode. It can also be observed that RL-LSTM with TL has less bouncing in prediction accuracy during the exploration time than the RL-LSTM without TL. This indicates the benefit of transferring knowledge in accelerating the search process and narrowing down the search space toward more optimal actions.

Fig. 9 demonstrates the Kernel Density Estimation (KD) of the achieved accuracy for each of the tested predictors. KD is defined in a non-parametric way and estimates the smoothed Probability Density Function (PDF) of a finite number of samples of an unknown density. It can be seen that TL-RL-LSTM has higher probability density (more users) across higher prediction accuracy values, and the smallest accuracy is at least 10% better than other predictors. Moreover, the distribution of the data for TL-RL-LSTM is also less dispersed than other predictors, meaning a more consistent accuracy performance.

Fig. 10 shows the average ML building model time per user for the different predictors. It can be observed that TL-RL-LSTM takes on average 176 minutes, approximately, to search the best architecture and train each user. However, in optimized deployments, this time would further decreased by parallelization. Alternatively, GS-LSTM, for smaller search space, takes on average 625 minutes to find the best architecture and train each user. Thus, TL-RL-LSTM converged to better neural architecture and respectively achieved higher prediction accuracy in much less time compared to GS-LSTM. In this context, a long ML building model time seems to be one of the drawbacks of the NN approaches, as both the RF and J48 algorithms took less than 1-hour on average to train each user. However, a longer time to build a NN prediction model in an offline manner aims the best prediction accuracy and less time to build the best NN prediction model when compared to other NN approaches.

Over the course of the experiments, it has been observed that the accuracy achieved by each user depends on the number of visited BSs by each user. Users connecting to fewer BSs tend to have more accurate prediction models. Fig. 11 presents users average accuracy for different predictors, grouped by

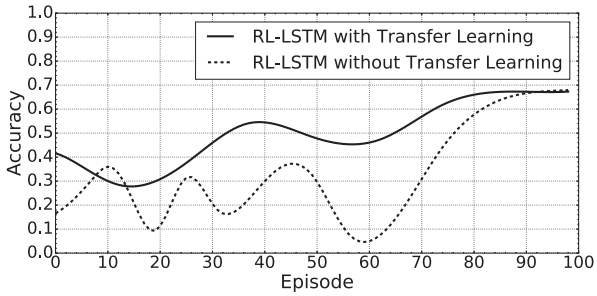


Fig. 8. Average prediction accuracy of 100 users with/without TL.

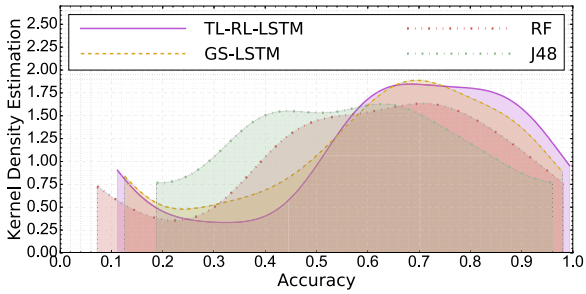


Fig. 9. Kernel density estimation accuracy.

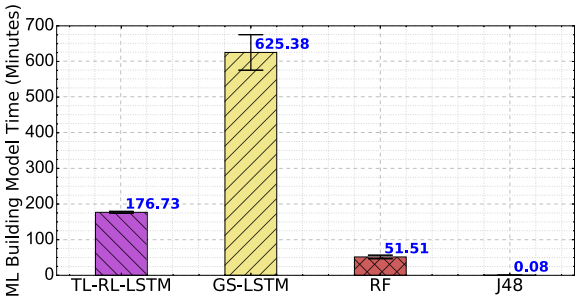


Fig. 10. Average model building time of 100 users.

user mobilities. We define different levels of user mobility by estimating each user's average number of visited BSs per day. From right to left of Fig. 11, groups are referred to very-high-mobility, high-mobility, medium-mobility, and low-mobility users. The very-high-mobility group contains 19-24 daily visited BSs, the high-mobility group contains 13-18 daily visited BSs, the medium-mobility group contains 7-12 daily visited BSs, and the low-mobility group contains 1-6 daily visited BSs. As it is shown, for each of the defined groups, TL-RL-LSTM predictor maintains the superior accuracy consistently. Further, we can see the trend of diminishing accuracy for very-high-mobility users. That is because the number of daily visited BSs grows, the error probability grows respectively. For the low-mobility users, the confidence interval of the predictors overlaps to a larger extent. This is because these users contain easier prediction scenarios. As it is discussed in Section III-C, users within the same mobility group have a similar correlation ratio which is defined as the ratio between the achieved prediction accuracy and the mean number of daily BSs. The correlation ratio of very-high-mobility users group is quite a small value since the achieved accuracy is low and the mean number of daily BSs is relatively high. Alternatively, the correlation

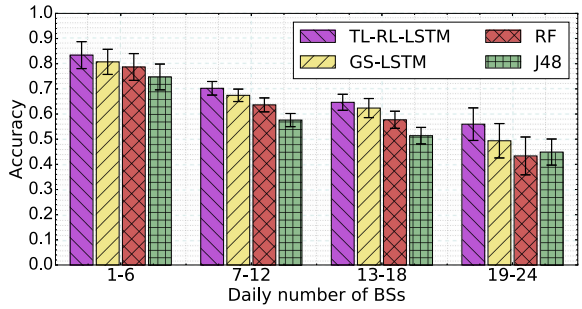


Fig. 11. Different predictors' achieved accuracy grouped by average number of daily visited BSs per User.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Number of UEs	20
Number of BSs	60
Macro Cell Transmission Power	46 dBm
UE Transmission Power	15 dBm
Propagation Loss Model	Nakagami
Simulated Time	100 Seconds
Number of Simulations	33

ratio of low-mobility users is quite a high value since the achieved accuracy is very high, and the mean number of daily BSs is low. The correlation ratios of other groups of high-mobility and medium-mobility are smoother values. Overall, it can be concluded that for users with medium and high mobility patterns, the proposed predictor performs much better than others, and their prediction accuracy is more similar to the average accuracy of total users introduced in Fig. 10.

D. Handover Optimization Evaluation

We implemented the RL-HEC algorithm in the NS-3.30[2] network simulator. RL-HEC was implemented on the LTE stack of the simulator accordingly to the Orange Dataset scenario, where 33 simulations have been performed using the parameters described in Table II and the scenario of Fig. 3. In each simulation, the random seed of the simulator was varied, and different users from the dataset were chosen to populate the scenario. Results are shown with a confidence interval of 95%. Each user in the scenario is equipped with a UDP-based application. Users move according to real-world traces, and each BS in the scenario is assigned with one device generating a constant bit rate traffic of 1 Mbps.

For comparison purposes, we also implemented three other state-of-the-art HO algorithms to test against RL-HEC. The algorithms are the following: *(i):* PRED [20] is position prediction-based HO algorithm based on the Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ) and some UE parameters like moving direction and the position inside the BS used as HO decision criteria. *(ii):* Received Signal Strength Indication (RSSI)-based [38], a standard HO algorithm based on signal events, meaning the events when the serving BS's signal quality drops below a threshold,

[2] https://nsnam.org

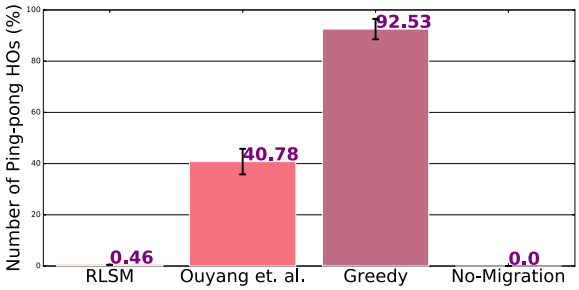


Fig. 12. Percentage of Ping-Pong HOs for each algorithm.

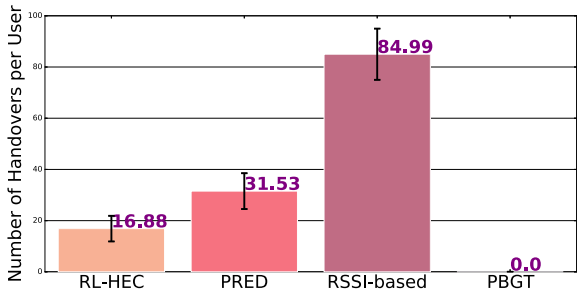


Fig. 13. Number of HOs for each algorithm.

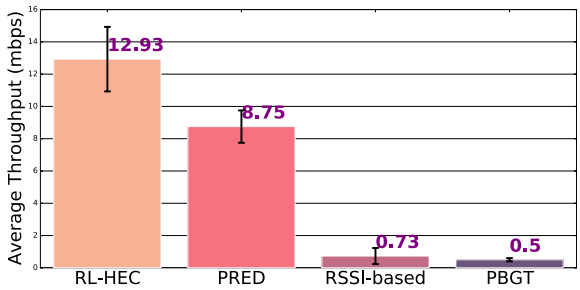


Fig. 14. Average network throughput for each algorithm.

and the events when a neighbor BS's signal quality is a certain threshold above the serving BS's one. This algorithm uses signal quality in its decision, making it more sensitive to interference and noise fluctuations. *(iii):* the strongest BS algorithm Power Budget (PBGT) [38] uses a threshold value, i.e., a HO is made if a neighbor BS's signal strength becomes larger than the serving BS's one plus a threshold. This makes this algorithm more robust to ping-pong HOs and interference as well, but it can cause users to stay in overloaded macrocells.

Fig. 12 shows the percentage of ping-pong HOs made in comparison with the total number of HOs in the simulations. Results were averaged across all 33 simulations and are shown with a confidence interval of 95%. Note that the amounts shown are cumulative across all 20 devices in the scenario. As previously defined, we consider a ping-pong HO as a disconnection and reconnection to a BS within 4 seconds [32]. We can see that RL-HEC achieves a near-zero number of ping-pong HOs, compared to the average of 40% and 92% of the PRED and RSSI-based algorithms, respectively. PRED maintains a relatively low number of ping-pong HOs, about 30 per user device, compared to the RSSI-based algorithm. This is because the PRED algorithm's predictive approach uses parameters such as moving direction and position inside the coverage area, which is not enough to predict and avoid the occurrences of ping-pong HOs. The RSSI-based algorithm performs a much larger number of ping-pong HOs because it is more sensible to signal fluctuations, especially when the coverage area of neighbor BSs and the serving BS overlap. The PBGT algorithm did not cause ping-pong HOs at all in the simulations. This is because it is executed with a significant hysteresis value that only allows a HO to happen, if the target BS's signals strength is a threshold above the serving one's.

The number of ping-pong HOs for each algorithm is proportional to the total number of HOs executed. Fig. 13 shows the raw number of HOs with different algorithms. RL-HEC caused about 16 HOs per node during 100 seconds of simulation, considering that, on average, each user in the simulation passes through the coverage area of 66 BSs. However, this does not mean that coverage areas do not overlap. RL-HEC tends to maximize the staying time of a user in a certain BS, i.e., the time the user remains connected to it, and then performs a HO. Both RL-HEC and PBGT performed less than 20 HOs per node. This is because in the decision phase of RL-HEC and PBGT the UE often only performs a HO when its current BS is unavailable, avoiding fluctuations. RL-HEC's mobility prediction makes the BS choice more reliable and connections more stable. In the PBGT algorithm, connections are stable because most users are bound to the BSs with the highest transmission power, ultimately increasing congestion levels and compromising QoS. This showcases how in scenarios with BSs of different transmission power value, traditional HO algorithms may fall short in performance. Each node under PRED performed on average one HO every 4 seconds. In the case of the RSSI-based algorithm, which is more sensitive to signal fluctuations, an excessive number of HOs is performed. In our experiments, with the RSSI-based algorithm, individual users perform one HO every 2 seconds.

Fig. 14 shows the impact of each HO algorithm in raw user throughput. Our evaluation methodology for throughput consists of empirical measurements. A variable bitrate UDP application is installed on each user device sending data from the UE to a remote host at the core of the network and measuring the end-to-end throughput. Transmission power for the BSs is set to 46 dBm using Multiple-Input Multiple-Output (MIMO) transmission mode. It is important to notice that users in this scenario do not have obstacles to their respective BSs, always maintaining a clear line-of-sight. User throughput will be heavily affected if the user is not connected to the most appropriate BS, due to factors like Signal to Noise Ratio (SINR), user movement, and interference from neighbor BSs. We can see that RL-HEC achieves a throughput of 12.9 Mbps, compared to 8.75 Mbps for the PRED algorithm, 0.73 Mbps for the RSSI-based, and 0.5 Mbps for the PBGT algorithm. This highlights that RL-HEC choses the best BSs most frequently compared to the other tested algorithms. PRED's throughput comes closer to it, however, at higher costs in terms of ping-pong HOs. The raw throughput achieved in the RSSI-based simulation is several times lower than the one of RL-HEC, due to excessive and sub-optimal HO executions.

TABLE III
SIMULATION PARAMETERS

Parameter	Value
Number of UEs	20
Number of Macro s	60
Macro Cell Transmission Power	46 dBm
User Device Transmission Power	15 dBm
Propagation Loss Model	Nakagami
Simulated Time	100 seconds
Service Modeled	Augmented Reality
Number of Edge Servers	60
Number of Simulations	33

E. Service Migration Evaluation

We now evaluate the performance of the RLSM algorithm proposed in this work. In this context, we consider each BS to be associated with an edge datacenter capable of running cloud-based services for users in the scenario. The algorithm takes advantage of learning models at the edge of the network to optimize the services being consumed at the network's edge. The architecture can take advantage of the user mobility prediction scheme to perform service migrations proactively to the next edge server the user shall connect to. Modern low-latency applications are especially sensitive to user mobility in this manner since the occurrence of HOs may cause the services being consumed to be rerouted and increase the end-to-end latency. Thus, edge services should always be close to their user to some extent. Note that simply transferring the service to the closest edge server ignores the dynamic nature of edge-enabled networks, such as how the available resources of the servers may be reserved by other users at the moment of the migration request.

Users in the simulated scenario consume a cloud-based application. For this work, we choose an Augmented Reality (AR) application with the requirements as defined by Lau *et al.* [39]. This application was selected as it is one of the emerging applications in intelligent MEC scenarios. It will benefit from the presence of ML models at the edge. Its requirements are defined as low latency (generally agreed upon at 10 ms maximum), high bandwidth, and moderate to high priority when compared with less demanding applications.

Many works in the recent literature tackle these problems, some of which consider mobility prediction. However, we found no other work that uses a next BS prediction for service migration optimization. As seen in Section II, the majority of state-of-the-art works do not consider the resources at the target servers for their decisions, often resulting in migration failures and increased latency. To compare RLSM with other works, we chose three other algorithms found in the literature: *(i):* Ouyang *et al.* [16] to represent the state-of-the-art in our simulations proposes "Follow me at the Edge," a baseline service migration scheme for edge-enabled networks which applies a Markov approximation to find a near-optimal behavior. *(ii)* A greedy approach, in which the network tries to always keep services in the edge server closest to the user consuming them. However, this approach increases the chance of migration failures and dramatically decreases the number of available computing resources at the network's edge. Furthermore, at last,

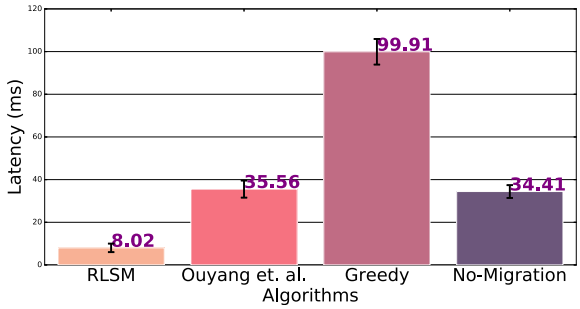


Fig. 15. Average service latency.

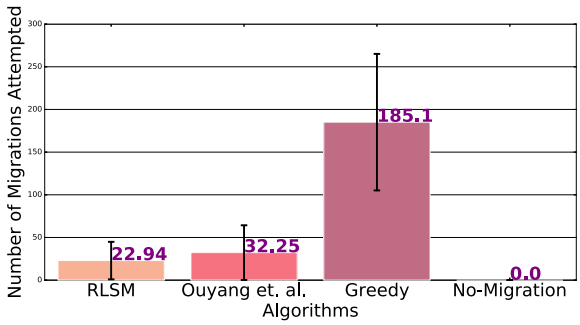


Fig. 16. Number of service migrations attempted.

(iii) a No Migration approach in which services are allocated to the edge, fog, and cloud servers at the beginning of the simulation and are not migrated for the remaining of the simulation. Simulations are conducted 33 times, with different random seeds and users' set from the dataset present in the scenario. Table III shows the main parameters used in the simulations and results are shown with a confidence interval of 95%.

In Fig. 15, we see the average latency across all users in the network for the service being tested. When considering the application's latency requirements, RLSM is the only algorithm that meets the application requirement threshold with average latency below 10 ms. This is because, most times, all users in the simulation are served by an edge or fog server. The same approach is attempted by the greedy approach, lacking proper resource management, causing many migration failures. In the simulation for the algorithm by Ouyang *et al.*, the end-to-end latency for the application is 35 ms on average, with an even larger average latency than the scenario where no migration occurs. This is because, in the No-Migration approach, users remain served at the server they are first allocated to, thus having a portion of users with low mobility remain connected to their closest edge server. In our scenario, the latency to reach a fog server is in the order of 10 ms, and the latency to reach an edge server is in the order of 1 ms. The greedy algorithm has the worst performance in terms of latency, as the many migration failures caused by the excessive number of migrations significantly deplete network resources and performance.

The number of migrations performed by each algorithm may also influence its performance, as shown in Fig. 16. In the simulated scenario, RLSM performed, on average, about 60 migrations per simulation on the total, against 90, 105, and 0 for the algorithm by Ouyang *et al.*, the greedy algorithm, and

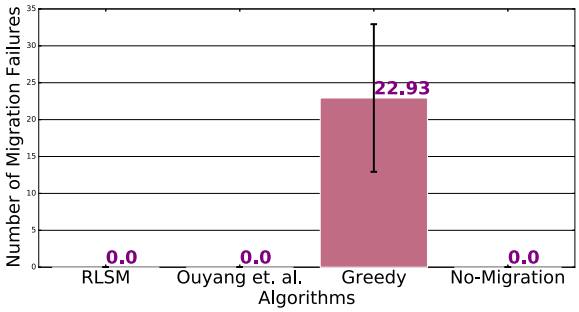


Fig. 17. Number of service migration failures.

the No-Migration algorithm, respectively. Since migrations in RLSM are made proactively, migrations tend to follow user movement and are more robust to signal fluctuations. The algorithm by Ouyang *et al.* and the greedy approach make a reactive migration after the user moves to a new area, which means that they can not reserve resources from the target servers proactively and may have to deal with migration failures, increasing the end-to-end latency of the applications. The No-Migration approach is configured not to perform any migrations during the course of the simulation.

One important resource management metric is the migration failure rate. We define a migration failure when migration is requested to a server with the necessary resources to support the applications, thus requiring another server to be chosen. Fig. 17 shows the number of migration failures, on average, for each algorithm. The no-migration approach did not perform any migrations, so it has no failures. The greedy strategy has the highest number of failures due to the lack of mobility prediction and resource awareness. RLSM did not cause any migration failures in the course of the simulations. This is because a resource check precedes every migration decision. Thus, migrations are only made to servers that can receive the service with ease. In terms of failures, Ouyang *et al.* 's algorithm caused fewer failures than the greedy approach, as expected, but still many more than RLSM.

VIII. Conclusion

User mobility awareness plays an essential role in enhancing network performance. In this article, we tackled the problem how user mobility prediction can be used to deploy NN models to optimize network performance. We designed an RL method to automate the architecture search for the LSTM networks with fast convergence rate. We validated our ideas on a real-world large scale anonymized dataset collected from a telecommunication network operator. Experiment results show that our predictor delivers better accuracy over state-of-the-art works. Moreover, we designed and implemented a novel HO algorithm and service migration scheme that benefit from mobility prediction. Simulation results show that the proposed solutions could reduce ping-pong HO rates to almost zero while increasing measured network throughput by 1.5 times compared to state-of-the-art solutions and lead to a much lower number of migration attempts and failures.

References

[1] J. K. Lee, Y. S. Jeong, and J. H. Park, "S-ITSF: A service based intelligent transportation system framework for smart accident management," *Hum.-Centric Comput. Inf. Sci.*, vol. 5, no. 1, pp. 34–42, 2015.

[2] W. Zheng, X. Huang, and Y. Li, "Understanding the tourist mobility using gps: Where is the next place?," *Tourism Manage.*, vol. 59, pp. 267–280, 2017.

[3] A. Mohamed, O. Onireti, S. A. Hoseinitabatabaei, M. Imran, A. Imran, and R. Tafazolli, "Mobility prediction for handover management in cellular networks with control/data separation," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 3939–3944.

[4] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Graph convolutional recurrent neural network: Data-driven traffic forecasting," *CoRR*, 2017, *arXiv:1707.01926*.

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

[6] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. Jul., pp. 1633–1685, 2009.

[7] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, no. 7196, pp. 779–782, 2008.

[8] G. Chen, A. C. Viana, M. Fiore, and C. Sarraute, "Complete trajectory reconstruction from sparse mobile phone data," *EPJ Data Sci.*, vol. 8, no. 1, p. 30, 2019.

[9] W. Zhang *et al.*, "A survey on decision making for task migration in mobile cloud environments," *Pers. Ubiquitous Comput.*, vol. 20, no. 3, pp. 295–309, 2016.

[10] H. Jing *et al.*, "LSTM-based service migration for pervasive cloud computing," in *Proc. IEEE Int. Conf. Internet Things IEEE Green Comput. Commun. IEEE Cyber, Physical Soc. Comput. IEEE Smart Data*, 2018, pp. 1835–1840.

[11] X. Yu, M. Guan, M. Liao, and X. Fan, "Pre-migration of vehicle to network services based on priority in mobile edge computing," *IEEE Access*, vol. 7, pp. 3722–3730, 2019.

[12] W. Zhang *et al.*, "Resource requests prediction in the cloud computing environment with a deep belief network," *Softw. - Pract. Experience*, vol. 47, no. 3, pp. 473–488, 2017.

[13] Z. Liang, Y. Liu, T. M. Lok, and K. Huang, "Multi-cell mobile edge computing: Joint service migration and resource allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 5898–5912, Sep. 2021.

[14] W. Zhang, S. Tan, Q. Lu, X. Liu, and W. Gong, "A genetic-algorithm-based approach for task migration in pervasive clouds," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 8, 2015, Art. no. 463230.

[15] C. Li, L. Zhu, W. Li, and Y. Luo, "Joint edge caching and dynamic service migration in SDN based mobile edge computing," *J. Netw. Comput. Appl.*, vol. 177, no. Jul. 2020, 2021, Art. no. 102966.

[16] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[17] Z. Gao, J. Meng, Q. Wang, and Y. Yang, "Service migration for deadline-varying user-generated data in mobile edge-clouds," in *Proc. IEEE World Congr. Serv., Services*, 2018, pp. 53–54.

[18] X. Xu, X. Tang, Z. Sun, X. Tao, and P. Zhang, "Delay-oriented cross-tier handover optimization in ultra-dense heterogeneous networks," *IEEE Access*, vol. 7, pp. 21 769–21 776, 2019.

[19] F. Gong, Z. Sun, X. Xu, Z. Sun, and X. Tang, "Cross-tier handover decision optimization with stochastic based analytical results for 5G heterogeneous ultra-dense networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2018, pp. 1–6.

[20] M. Mandour, F. Gebali, A. D. Elbayoumy, G. M. A. Hamid, and A. Abdelaziz, "Handover optimization and user mobility prediction in lte femtocells network," in *Proc. IEEE Int. Conf. Consum. Electron.*, 2019, pp. 1–6.

[21] R. Ma, J. Cao, D. Feng, H. Li, and S. He, "FTGPHA: Fixed-trajectory group pre-handover authentication mechanism for mobile relays in 5G high-speed rail networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2126–2140, Feb. 2020.

[22] S. Lefevre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *Robomech J.*, vol. 1, no. 1, pp. 1–14.

[23] C. Barrios, Y. Motai, and D. Huston, "Trajectory estimations using smartphones," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7901–7910, Dec. 2015.

[24] G. Xie, H. Gao, L. Qian, B. Huang, K. Li, and J. Wang, "Vehicle trajectory prediction by integrating physics- and maneuver-based approaches using interactive multiple models," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5999–6008, Jul. 2018.

[25] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 4363–4369.
[26] J. Chung, C. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, 2014, *arXiv:1412.3555*.
[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
[28] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer, "Generalizable intention prediction of human drivers at intersections," in *Proc. IEEE Intell. Veh. Symp.*, 2017, pp. 1665–1670.
[29] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016.
[30] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 281–305, 2012.
[31] 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," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[32] L. Tartarini *et al.*, "Software-defined handover decision engine for heterogeneous cloud radio access networks," *Comput. Commun.*, vol. 115, pp. 21–34, 2018.
[33] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.
[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.
[35] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[36] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[37] T. Chen, I. J. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *CoRR*, 2016, *arXiv:1511.05641*.
[38] 3GPP, "Evolved universal terrestrial radio access (e-utra); radio resource835control (rrc); protocol specification, 3gpp ts 36.331 v9.4.0 (2010-09)," Evolved Universal Terrestrial Radio Access (E-UTRA);Radio Resource837-Control (RRC);Protocol specification(Release 9), 2011.
[39] B. P. L. Lau *et al.*, "A survey of data fusion in smart city applications," *Inf. Fusion*, vol. 52, pp. 357–374, 2019.

Zhongliang Zhao received the Ph.D. degree from the University of Bern, Bern, Switzerland, in 2014. Since 2019, he has been an Associate Professor with Beihang University, Beijing, China, and a Senior Researcher with the University of Bern. His research interests include machine learning, edge computing, and UAV ad-hoc networks.

Negar Emami received the master's degree in telecommunications engineering from the Politecnico di Milano, Milan, Italy. She is currently working toward the Ph.D. degree with the Department of Computer Science, University of Bern, Bern, Switzerland. Her research interests include modeling mobility data for network management and enhancing location-based applications.

Hugo Santos received the graduation degree in computer engineering and the master's degree in electrical enginering from the Federal University of Pará, Belém, Brazil. He is currently under a jointly Ph.D. supervision with the University of Bern, Bern, Switzerland. His research interests include cloud, fog and edge computing, vehicular UAV networks, mobility, multimedia, and quality of experience.

Lucas Pacheco received the undergraduate and master's degrees in electrical engineering from the Federal University of Pará, Belém, Brazil. Since 2020, he has been working toward the Ph.D. degree in computer science with the University of Bern, Bern, Switzerland. His research interests include mobility management, prediction and management of cloud resources, and the impact of urban mobility on cloud, and edge services.

Mostafa Karimzadeh received the Ph.D. degree in computer science from the University of Bern, Bern, Switzerland. His research interests include computer and communication networks, machine learning, and deep learning optimization for network performance.

Torsten Braun (Senior Member, IEEE) received the Ph.D. degree from the University of Karlsruhe, Karlsruhe, Germany, in 1993. From 1994 to 1995, he was a Guest Scientist with INRIA Sophia-Antipolis, France. From 1995 to 1997, he was a Project Leader and a Senior Consultant with IBM European Networking Centre, Heidelberg, Germany. Since 1998, he has been a Full Professor of computer science with the University of Bern, Bern, Switzerland.

Arnaud Braud is currently an Architect and Data Scientist with Orange Laboratories involved in AI powered digital marketplaces research studies and also 5G core data analytics functions assessment. He is also contributing to the European Union Gaia-X initiative.

Benoit Radier received the electrical engineering Diploma from the Institut Supérieur de l'Electronique et du Numérique, Brest, France, and the Doctoral degree in computer science from the Université Pierre et Marie Curie, Paris, France, in 2009. He is currently a Research Engineer in informatics and telecommunication. In 2000, he joined France Télécom Research and Development Lannion, France. Since 2007, his research interests has been include autonomic networking, generic autonomic network architecture, context awareness, and knowledge plane.

Philippe Tamagnan is currently an Architect with Orange Laboratories involved in AI powered 5G core with data analytic functions research studies and assessment. He is also contributing to the 3GPP.