# Deep Learning Techniques for Mobility Prediction and Management in Mobile Networks

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

**Negar Emami**

von Teheran, Iran

Leiter der Arbeit:
Professor Dr. Torsten Braun

Institut für Informatik

# Deep Learning Techniques for Mobility Prediction and Management in Mobile Networks

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

**Negar Emami**

von Teheran, Iran

Leiter der Arbeit:
Professor Dr. Torsten Braun

Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 27 September, 2023

Der Dekan:
Prof. Dr. Marco Herwegh

*To my father 'Baba Isa', whose consistent support and boundless knowledge have been a constant source of inspiration and motivation throughout my doctoral journey.*

*"The true delight is in the finding out rather than in the knowing."*

Isaac Asimov

# *Abstract*

Trajectory prediction is an important research topic in modern mobile networks (e.g., 5G and beyond 5G) to enhance the network quality of service by accurately predicting the future locations of mobile users, such as pedestrians and vehicles, based on their past mobility patterns. A trajectory is defined as the sequence of locations the user visits over time.

The primary objective of this thesis is to improve the modeling of mobility data and establish personalized, scalable, collective-intelligent, distributed, and strategic trajectory prediction techniques that can effectively adapt to the dynamics of urban environments in order to facilitate the optimal delivery of mobility-aware network services. Our proposed approaches aim to increase the accuracy of trajectory prediction while minimizing communication and computational costs leading to more efficient mobile networks.

The thesis begins by introducing a personalized trajectory prediction technique using deep learning and reinforcement learning. It adapts the neural network architecture to capture the distinct characteristics of mobile users' data. Furthermore, it introduces advanced anticipatory handover management and dynamic service migration techniques that optimize network management using our high-performance trajectory predictor. This approach ensures seamless connectivity and proactively migrates network services, enhancing the quality of service in dense wireless networks.

The second contribution of the thesis introduces cluster-level prediction to extend the reinforcement learning-based trajectory prediction, addressing scalability challenges in large-scale networks. Cluster-level trajectory prediction leverages users' similarities within clusters to train only a few representatives. This enables efficient transfer learning of pre-trained mobility models and reduces computational overhead enhancing the network scalability.

The third contribution proposes a collaborative social-aware multi-agent trajectory prediction technique that accounts for the interactions between multiple intra-cluster agents in a dynamic urban environment, increasing the prediction accuracy but decreasing the algorithm complexity and computational resource usage.

The fourth contribution proposes a federated learning-driven multi-agent trajectory prediction technique that leverages the collaborative power of multiple local data sources in a decentralized manner to enhance user privacy and improve the accuracy of trajectory prediction while jointly minimizing computational and communication costs.

The fifth contribution proposes a game theoretic non-cooperative multi-agent prediction technique that considers the strategic behaviors among competitive inter-cluster mobile users.

The proposed approaches are evaluated on small-scale and large-scale location-based mobility datasets, where locations could be GPS coordinates or cellular base station IDs. Our experiments demonstrate that our proposed approaches outperform state-of-the-art trajectory prediction methods making significant contributions to the field of mobile networks.

# *Acknowledgements*

I am pleased to present my doctoral thesis, which was carried out at the University of Bern under the Swiss National Science Foundation (SNSF) project "Intelligent Mobility Services". It has been a long and challenging journey, and I would like to thank all the people who have contributed to my academic achievements and made this possible.

I would like to express my deepest gratitude to my supervisor, Prof. Dr. Torsten Braun, for his invaluable support and guidance throughout my Ph.D. journey. I am grateful to him for affording me the opportunity to join the Communications and Distributed Systems (CDS) group in the Institute of Informatics at the University of Bern and pursue my Ph.D.. I am also thankful for having the opportunity to participate in diverse and intellectually stimulating activities, such as summer schools, winter schools, seminars, and conferences. These activities have not only contributed significantly to my academic growth but have also provided me with the priceless opportunity to network and interact with esteemed researchers and scientists from various parts of the globe. Dear Prof. Braun, I appreciate your guidance in helping me become an independent researcher and follow my own ideas.

I am grateful to SNSF for providing me with the opportunity to work in a highly qualified environment at the University of Bern and for supporting the national project "Intelligent Mobility Services". I also thank Orange telecommunication company in France, for providing us with the confidential mobility dataset that we used throughout this thesis.

I am thankful to Prof. Dr. Zhongliang Zhao and Dr. Antonio di Maio for their valuable advice and corrections during my doctoral studies. I especially thank my CDS colleagues, including Maria, Eric, Hugo, Dimitrios, Diego, Lucas, Tofumni, Allison, and Eirini for the memorable conversations and moments we shared over the last four years. I also thank my old Iranian friends for their support and encouragement throughout this journey.

I want to extend my thanks to CDS secretaries, Daniela Schroth and Priska Grunder, for their infinite assistance with administrative procedures during the completion of my Ph.D.. I also thank Dr. Peppo Brambilla, the computer science institute's system administrator, for his infinite help.

I am deeply grateful to my parents, maman Rafi and baba Isa, as well as my brother Nima, for their absolute love and support. Their encouragement has been a constant source of motivation, and they have been there for me during the most challenging times. Finally, I would like to thank my husband and best friend, Luca, for his unconditional support, love, and belief in me, especially during the times when I had lost faith in myself. I cannot thank him enough for his patience and understanding, allowing me to pursue my dreams. My dear family, you have always been my pillars of strength, and I could not have achieved this success without you.

Negar Emami
Bern, 27 September, 2023

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AI** | **Artificial Intelligence** |
| **AMF** | **Access and Mobility Management Function** |
| **ANN** | **Artificial Neural Network** |
| **AR** | **Augmented Reality** |
| **BS** | **Base Station** |
| **CNN** | **Convolutional Neural Network** |
| **DL** | **Deep Learning** |
| **FL** | **Federated Learning** |
| **GAN** | **Generative Adversarial Network** |
| **GPS** | **Global Positioning System** |
| **GRU** | **Gated Recurrent Unit** |
| **GS** | **Grid Search** |
| **GT** | **Game Theory** |
| **Ho** | **Handover** |
| **HO** | **HyperOpt** |
| **IoT** | **Internet of Things** |
| **ITS** | **Intelligent Transportation System** |
| **KF** | **Kalman Filter** |
| **LBS** | **Location-Based Service** |
| **LSTM** | **Long Short Term Memory** |
| **MDP** | **Markov Decision Process** |
| **MEC** | **Multi-Access Edge Computing** |
| **MIMO** | **Multiple Input Multiple Output** |
| **ML** | **Machine Learning** |
| **MME** | **Mobility Management Entity** |
| **NAS** | **Neural Architecture Search** |
| **NLP** | **Natural Language Processing** |
| **NN** | **Neural Network** |
| **QoE** | **Quality of Experience** |
| **QoS** | **Quality of Service** |
| **RF** | **Random Forest** |
| **RL** | **Reinforcement Learning** |

| | |
|---|---|
| **RL-HEC** | **Reinforcement Learning-based Handover for Edge Computing** |
| **RL-SM** | **Reinforcement Learning-based Service Migration** |
| **RNN** | **Recurrent Neural Network** |
| **RS** | **Random Search** |
| **RSSI** | **Received Signal Strength Indication** |
| **SDN** | **Service Defined Network** |
| **SGD** | **Stochastic Gradient Descent** |
| **TF** | **Transformer** |
| **TL** | **Transfer Learning** |
| **TP** | **Trajectory Prediction** |
| **UE** | **User Equipment** |
| **XR** | **Extended Reality** |

# Chapter 1

# Introduction

The primary objective of this thesis is to explore the potential of the historical location and mobility data in predicting future trajectories of mobile individuals or groups, including both pedestrians and vehicles. The objective is to enhance the performance of mobility management units and handover procedures in modern wireless networks, such as 5G and beyond-5G systems. These improvements will facilitate multi-edge computing, enabling closer data migration to edge servers and end mobile users, thus creating a more efficient and robust wireless communication environment.

Mobility trace data are considered as time-series data that have been collected at different points in time such as Global Positioning System (GPS) coordinates, WiFi access points, cellular base stations, or location tags from social media. A mobility trace, or a series of visited locations, can be viewed as a trajectory. Time-series prediction can be generalized as a process that extracts useful information from historical records and estimates future values[25]. Therefore, a mobility or trajectory predictor models users' movements and anticipates their likely future locations.

This research seeks to explore new ways of leveraging location and mobility data to better understand and predict mobility patterns and mutual interactions among multiple interfacing users in an urban environment. We aim to boost the effectiveness of services that depend on proactive and anticipatory mobility predictions while reducing both the computational and communication complexity. Our focus is to improve existing mobility predictors' *performance* in terms of ***reliability***, ***speed of convergence***, ***scalability***, ***computational efficiency***, ***privacy***, ***communication network adaptivity***, *and* ***strategic robustness*** to various urban environment scenarios and wireless network conditions. In this thesis, we identify the challenges associated with current machine learning-based trajectory predictors concerning these metrics. Subsequently, we present five trajectory predictors as our contributions, each designed to enhance the performance of the aforementioned metrics while considering distinct scenarios, including isolated-agent prediction, multi-agent prediction, centralized training, distributed training, cooperative interactions, and non-cooperative interactions among multiple mobile users in a dynamic urban scenario.

This chapter sets the foundation for the rest of the thesis, providing an overview of the research questions and the contributions that it makes toward improving mobility and Trajectory Prediction (TP) schemes. In the beginning, it provides an introduction to the concept of mobility

prediction, highlighting its relevance in the context of location-based and proactive mobility services. Afterwards, it discusses the fundamental limitations of TP models, identifying the gaps that exist in current literature research. The chapter then goes on to outline the contributions that this thesis makes toward addressing these limitations, and provides an overview of the subsequent chapters.

## 1.1 Overview

The advent of next-generation wireless networks demands self-organization, cost-effectiveness, efficiency, and a high Quality of Experience (QoE) for consumers. While 5G networks are already being deployed worldwide, the evolution beyond 5G, including 6G, must seamlessly handle user mobility to meet the increasing demands of modern connectivity [32]. User mobility serves as a defining characteristic of contemporary networks, encompassing crucial management aspects such as handovers, service disruptions, network congestion, load balancing, and dynamic resource allocation. It plays a pivotal role in ensuring uninterrupted connectivity, optimizing network performance, and accommodating the dynamic needs and preferences of users as they move within the network coverage area. In the pursuit of next-generation networks, proactive mobility management strategies leveraging Artificial Intelligence (AI) for forecasting and handling user mobility become indispensable [85]. Adaptive and anticipatory network management enhances operators to efficiently allocate resources, optimize network costs, and enhance the user experience. Mobility prediction, in particular, emerges as a vital component for improving the Quality of Service (QoS) of network applications [99]. Accurate prediction of user movement patterns enables network management systems to proactively manage handovers, migrate services, and allocate network resources to meet user demands, resulting in a seamless and enhanced user experience [104]. By integrating advanced mobility prediction techniques, next-generation networks can optimize network operations and ensure superior QoS and QoE for users.

On the other hand, trajectory and mobility prediction play a critical role in enhancing safety in urban environments, as well as enabling efficient autonomous driving and robot-to-human interaction systems [46, 73]. By accurately forecasting future paths of moving objects, such as vehicles or pedestrians, autonomous vehicles can navigate through traffic, avoiding obstacles and hazards. This ensures the safety of passengers, pedestrians, and other road users [63]. Furthermore, mobility prediction is essential in the development of advanced traffic management systems, which can improve traffic flow and reduce congestion in urban areas [75]. This optimization of travel time and reduction of accident risk can greatly benefit urban areas. Predicting mobility patterns and trajectories of mobile users is a vital component in the fields of urban planning, smart cities, Intelligent Transportation Systems (ITSs), emergency applications, and rescue operations [14].

The rapid advancement of Location-Based Services (LBSs), GPS, and other technologies such as smartphones, social network applications, wireless access points, cellular base stations, and satellites have generated an enormous amount of location data [7, 15]. This data has become a valuable resource for understanding human motion behaviors in different contexts, such as urban mobility, congestion, and social interactions. However, the sheer volume and complexity of location data present a significant challenge for traditional statistical modeling and analysis

techniques. In recent years, Machine Learning (ML), Deep Learning (DL), and Artificial Neural Networks (ANNs) have emerged as powerful tools to learn features from location data and predict human behavior patterns [50, 61]. These techniques can help develop accurate and efficient prediction models, leading to more effective and personalized LBSs.

However, despite the promising results of ML and ANNs in learning location data features, spatiotemporal dependencies, and user trajectories, several research challenges remain to be addressed, which are the focus of this thesis. Specifically, this thesis proposes high-performance mobility and trajectory prediction algorithms that surpass the state of the art, enhancing performance criteria such as reliability, convergence, scalability, computational efficiency, latency, and robustness. In the following two sections, we will provide an in-depth analysis of these challenges, and highlight and present the contributions of this thesis.

## 1.2 Problem Statement

This section provides an overview of the current shortcomings of existing Neural Network (NN)-based mobility prediction and management models and outlines the research questions that this thesis aims to address. We analyze the limitations of existing trajectory predictors across various scenarios, such as isolated-agent, multi-agent, centralized, distributed, cooperative, and non-cooperative trajectory prediction. These limitations primarily pertain to the reliability, robustness, scalability, computational and communication resource consumption, privacy, communication overhead, and adversarial interactions among mobile users in dynamic urban environments.

To comprehensively define the challenges of the state-of-the-art NN-based trajectory predictors, we begin by identifying the specific problems encountered in isolated-agent predictors. We then transition to multi-agent collaborative predictors, where the impact of users on each other becomes a critical factor to consider. Additionally, we explore the shift from centralized models to distributed models to ensure user data privacy. Furthermore, we analyze challenges within cooperative and non-cooperative social-aware trajectory predictors.

This thesis specifically emphasizes the identification of limitations and the proposal of solutions for cutting-edge NNs rather than non-NN machine learning models. NNs and deep learning have revolutionized time-series prediction tasks by their ability to capture complex non-linear relationships and dependencies within the sequential data [25]. Their neural architectures allow for the extraction of meaningful features at different levels of abstraction, enabling them to model both short-term and long-term dependencies effectively. Moreover, advancements in computational resources and parallel processing have facilitated the training of deep networks, enabling the exploration of larger and more sophisticated architectures. However, in the evaluation section of our study, we will conduct a comprehensive performance comparison between our proposed predictors and other NN and non-NN-based predictors.

### 1.2.1 Reliable and Computationally-Light Trajectory Prediction Gap in Mobility Management Techniques of Modern Wireless Networks

As stated, forecasting mobile users' behaviors and future trajectories is crucial for modern wireless networks to provide them with a high QoS for their applications [2]. Next-generation wireless

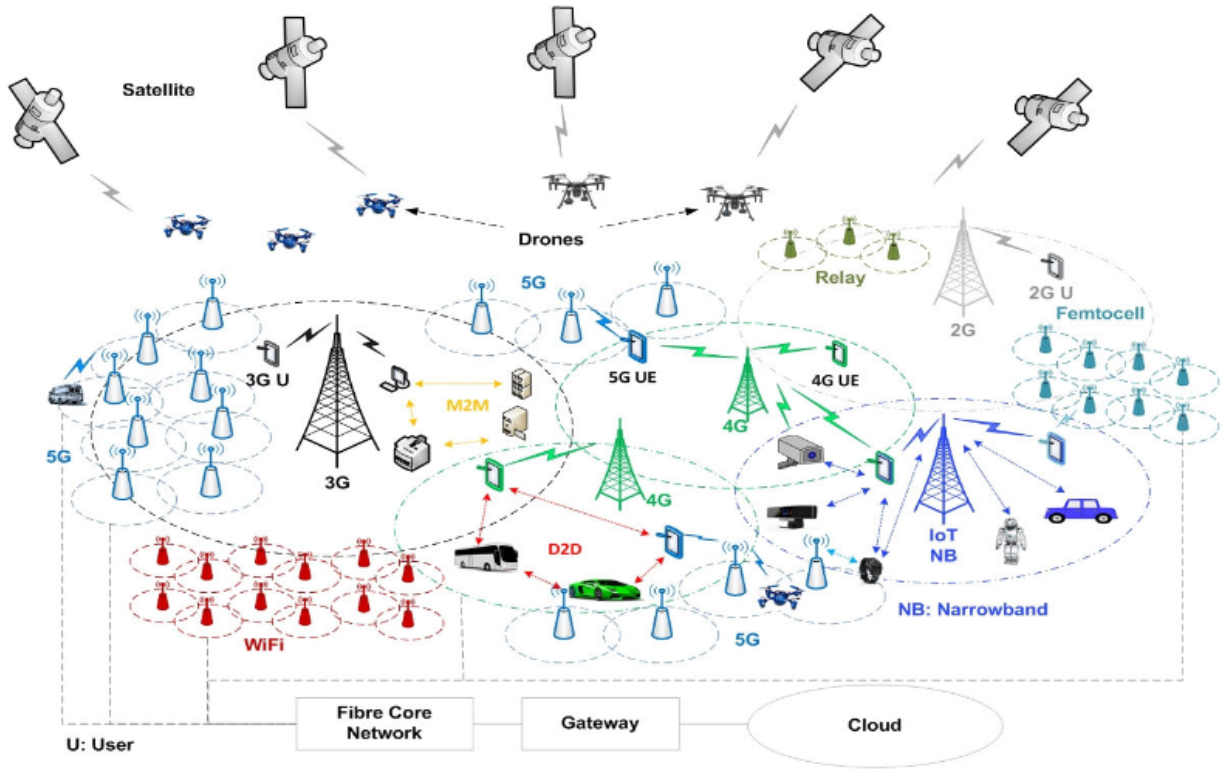



FIGURE 1.1: Mobility management in 5G ultra-dense networks. This image is sourced from the paper by Shayea et al. [78] published in IEEE Access that is licensed under a Creative Commons Attribution 4.0 license, which is different than the license of this thesis. For more information, see https://creativecommons.org/licenses/by/4.0/.

networks, such as 5G and beyond 5G, are characterized by more densely deployed base stations with smaller coverage areas, enabling finer-grained mobility information of users beyond their connection points in the network [78]. In addition to the sequence of small cells that users connect to in their trajectories, this information includes the frequency of connections and disconnections from base stations, such as handovers. Handover refers to the process in a mobile communication system where an active call or data session is transferred from one base station or cell to another as the mobile user moves through the wireless network. Therefore, the dense base station deployment in 5G and beyond results in more frequent handovers, which can cause a significant signaling overhead for the network, as well as frequent disruptions in the consumption of contents and services in case of handover failure. Figure 1.1 illustrates the crucial role of managing mobility in ultra-dense 5G networks containing various types of connections [78].

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 that 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 is closer to end-users, constituting another challenge from the mobility management standpoint, constituting the notion of Multi-Access Edge Computing (MEC) [38]. 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 [65].

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. In the context of wireless networks, the term "*service*", in *service migration*, refers to the various applications, functionalities, or tasks provided to users over the network. 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 [88]. However, since service migrations typically occur in a reactive manner after handover events, this can cause disconnections and interruptions to user services. In mobile networks, handover 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 [24]. 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.

In this direction, one of the major problems with state-of-the-art handover management works is the lack of resilient predictive schemes or mobility information incorporated into the decision-making process [93]. While some works attempt to address excessive and redundant handovers in small cell networks using mobility-aware algorithms, their solutions are offline and not fast-reactive enough for challenging real-time scenarios. Additionally, most models proposed for conventional wireless networks do not account for the higher number of handovers expected in new-generation ultra-dense small-cell modern networks, and may not be suitable for all scenarios.

Moving to the predictive service migration paradigm for an edge computing-enabled scenario, some works have proposed mobility prediction solutions that have certain limitations. Decision-making of service migration in MEC environments presents significant challenges, particularly when it comes to large-scale experiments. While existing solutions aim to reduce overall migration costs through specialized mobility models, these models are often heuristically designed and not feasible for real-time network management [95]. Moreover, most of the existing works propose an offline migration decision for services in mobile edge computing, which limits the system's ability to respond quickly to real-time changes.

As a result, developing high-performance mobility prediction mechanisms is essential to optimize mobility management units, including handover management and service migration, in wireless networks. Recently, Recurrent Neural Networks (RNNs) have emerged as groundbreaking models for time-series data analysis. Their unique architecture is specifically designed to handle sequential data by capturing dependencies and patterns over time. Unlike traditional feedforward neural networks, RNNs possess internal memory that allows them to retain information from previous time steps and incorporate it into the current prediction or decision-making

process. Moreover, RNN variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been introduced to address the vanishing gradient problem and improve the modeling of long-term dependencies. These architectures have further enhanced the performance of RNNs in processing sequential data.

However, the major challenge regarding existing RNN mobility predictors in literature is due to the inflexibility of their NN architectures, which are unable to adapt to new input data types or integrate new mobility features. Current TP models often rely on designing the NN architectures based on heuristics and experts' knowledge, which can be a time-consuming and error-prone process. Relying solely on experts' prior knowledge can be limiting and misleading in less-explored fields such as TP compared to well-established fields like image processing where high-performance models such as AlexNet, VGG, and ResNet, which are designed for well-explored data such as MNIST[1] and CIFAR10[2].

In this direction, Neural Architecture Search (NAS) approaches such as Hyperopt (a type of Auto-ML) have been proposed to automate the design of NN models to optimize the architecture for each user's unique movement patterns and mobility features [17]. Auto-ML[3], short for Automated Machine Learning, is a process that automates the selection and tuning of machine learning algorithms and hyperparameters. It employs optimization methods such as Bayesian optimization, grid search, random search, or evolutionary algorithms to search through the hyperparameter space and find optimal settings for a given ML model [65]. Auto-ML encompasses several techniques, including popular tools such as Hyperopt, Optuna, and Auto-sklearn.

NAS approaches aim to discover the optimal NN architecture for a given dataset's characteristics. The objective function of NAS is to identify the NN architecture that achieves the highest accuracy among a vast search space of possible architectures. However, this objective function is often non-convex due to the complex nature of the architecture search space, which involves discrete variables and non-linear relationships with performance metrics. Consequently, NAS becomes an NP-hard problem of integer non-linear programming, for which traditional optimization algorithms cannot guarantee an optimal solution in polynomial time.

On one hand, to model NAS in the field of TP, a few existing works use Auto-ML methods that can be computationally intensive and not efficient if the size of the search space grows due to the random search nature of such algorithms. On the other hand, some works employ brute force methods and grid search as NAS techniques, which are impractical to use in large-scale networks due to their exorbitant computational costs. Indeed, in certain cases, researchers may attempt to reduce the search space to make bruteforce methods more feasible. However, a drawback of such an approach is that it can compromise the guarantee of high accuracy, potentially leading to the exclusion of optimal models during the search process. Thus, keeping a balance between computational complexity, convergence rate, and accuracy becomes a significant challenge in NAS techniques.

Therefore, personalized NN architectures using more efficient and effective NAS techniques are needed to enhance the performance and reliability of the existing mobility predictors and offer

[1] http://yann.lecun.com/exdb/mnist/

[2] https://www.cs.toronto.edu/ kriz/cifar.html

[3] https://www.automl.org/automl/hpo-overview/

computationally-light solutions, leading to significant improvements in prediction-based mobility management services.

In the direction of the above limitations, we aim to address the following research questions.

- *RQ 1.1: How can we increase the reliability of current NAS techniques in the field of trajectory prediction, maintaining a desirable balance between the prediction accuracy and the NAS convergence rate?*
- *RQ 1.2: How can we enable a robust proactive handover mechanism with high QoS and service continuity for network applications like service migration, using NAS-enabled high-performance NNs?*

In Section 1.3.1, we outline our specific contributions in detail to address the aforementioned research questions.

### 1.2.2 Scalable and Convergent Individual-Agent Trajectory Prediction Gap in Centralized Large-Scale Mobility Networks

As we discussed in Section 1.2.1, designing a computationally-light NAS algorithm to personalize NNs for individual mobile users based on their unique data features can significantly enhance predictive accuracy while maintaining lower computational complexity compared to other NAS methods such as the popular AutoML. However, in the context of large-scale networks with thousand to millions of mobile users, it becomes impractical to train a separate personalized NN for each user. In large-scale mobility networks, computational complexity is a critical factor due to limitations on computational and communication resources within a centralized network. The use of multiple NAS-personalized NNs in such networks can be prohibitively expensive or even infeasible. Therefore, in this section, we aim to explore alternative approaches beyond individually training each user to come up with a scalable solution and cope with large-scale network requirements. We aim to apply scaling techniques that optimize the performance of the prediction process by efficiently utilizing computational resources while maintaining high prediction accuracy.

Moreover, although RNNs, and especially their variants LSTM models, have shown promising results in predicting mobility patterns and trajectories, they still have remarkable limitations. One of the main limitations of RNNs is their inability to capture long-term dependencies efficiently [58]. In other words, they may not be able to learn from past data that is far away from the current time step. This can result in the model not being able to make accurate predictions when the gap between the current and previous data is large. LSTM models were developed to address this limitation by introducing a memory cell that can selectively remember or forget information [96]. However, LSTM models still have challenges in capturing long-term dependencies in complex scenarios with high variability. Furthermore, RNNs and LSTM models may present high computational requirements as they process data in a sequential, rather than parallel, manner, and their implementation may demand a considerable amount of training data to attain high accuracy levels [65]. Such characteristics can render these models impractical for certain real-world scenarios. In this direction, in this section, we desire to explore alternative neural network architectures

for trajectory prediction that offer improved learning efficiency and convergent training through parallelization compared to traditional RNNs.

In the direction of the above limitations, we aim to address the following research questions.

- *RQ 2.1: How can we utilize alternative NN architectures for trajectory prediction instead of the commonly-used RNNs, considering their limitations of memory constraints, computational complexity, and difficulties in parallelization?*
- *RQ 2.2: How can we effectively scale NAS-based personalization in large-scale networks? What is the optimal tradeoff between computational resource consumption and prediction accuracy through scaling techniques?*

In Section 1.3.2, we outline our specific contributions in detail to address the aforementioned research questions.

### 1.2.3 Computationally-Efficient Multi-Agent Trajectory Prediction Gap in Centralized Social-Interactive Mobility Networks

In Section 1.2.1 and Section 1.2.2, we focus on modeling individual users, in small- and large-scale networks, respectively, where each agent is considered in isolation from the impact of its neighboring users' movements. However, in crowded public spaces, mobile users' movements are governed by social rules, such as estimating other users' mobility status, respecting their personal space, avoiding collisions, and coordinating group destinations. This implies that there is a strong interdependence among the mobility patterns and decisions of neighboring users. Understanding social interactions and spatio-temporal dependencies among mobile users' paths can significantly enhance the prediction of their complex motion behaviors. Socially-aware methods have demonstrated considerable improvement over socially-unaware methods in multi-agent scenarios where users are not acting in isolation. Therefore, in this section, we aim to explore social-aware collaborative trajectory prediction and enhance the performance of the existing works.

Recent research in trajectory prediction has undergone a shift from individual models to joint models, which leverage the mutual influence among individuals in a complex dynamic environment to form group intelligence. This approach allows individuals to learn from their own historical data and the historical data of their neighbors, resulting in improved prediction performance of the system. In recent years, data-driven social mobility predictors have become more popular compared to the previously proposed *Social-force* models, which rely on simple repulsive and attraction forces [34, 33] for group learning. ML and DL models have replaced these models due to their better modeling of sequential patterns and ability to achieve higher prediction accuracy with less computation. Rather than modeling kinetic forces and energy potentials as in social-force models, social-pooling [3, 31], attention [4, 74], and graph [94, 60] mechanisms complement NNs to share information about neighboring users' trajectories, capturing complex interactions in crowded environments.

Existing social-aware DL methods have demonstrated good performance in predicting social interactions among multiple mobile users. However, these algorithms suffer from the significant drawback of being computationally complex and resource-intensive. This is because the existing

algorithms require the trajectories of all users within a scene to be fed to the social predictor's encoders, leading to high computational costs and resource requirements. This can limit the scalability of these algorithms, especially in large-scale social trajectory prediction systems. Thus, in this section, our objective is to explore solutions that effectively capture user interactions from relevant users, those whose movements have a significant impact on each other's trajectories, rather than considering interactions from every individual.

Another important challenge in existing social-aware trajectory predictors is the lack of NN architecture adaptation, which can lead to suboptimal performance as the neural architecture may not capture the unique mobility patterns and behaviors of each user. Compared to isolated individual predictors, the NN inflexibility is even more pronounced in social-aware predictors where multiple users are merged into a shared neural layer to exchange their data. Therefore, in this section, we aim to broaden the scope of the discussed efficient NAS-enabled individual trajectory prediction to include social-aware trajectory prediction.

Additionally, while some works use CNNs for trajectory prediction, which show better efficiency than RNNs, they may not be fully suitable for sequential data. Therefore, we explore another efficient alternative, such as attention-based neural networks, which are well-suited for modeling time-series data.

To summarize, in this section, we aim to extend the scope of highly accurate, convergent, and scalable isolated-agent centralized trajectory prediction, as described in Section 1.2.1 and Section 1.2.2, to encompass multi-agent collaborative trajectory prediction. This expansion allows us to further optimize the trajectory prediction process by considering the interactions and dynamics among multiple agents. While centralized social-aware trajectory predictors have the potential to compromise privacy based on how they handle and store personal data, it is important to note that our current focus lies in addressing the problem of excessive computational resource consumption within existing state-of-the-art collaborative predictors. In subsequent stages of our research, we plan to incorporate privacy considerations by transitioning to decentralized models.

In the direction of the above limitations, we aim to address the following research questions.

- *RQ 3.1: How can we utilize alternative NN architectures for trajectory prediction instead of CNNs, to more effectively learn the sequential mobility data?*
- *RQ 3.2: How can we achieve a balance between reducing the computational complexity of modern trajectory predictors, which utilize social-aware techniques to capture user interdependencies, while ensuring that overall prediction accuracy is maintained?*
- *RQ 3.3: How can we integrate the NAS personalization paradigm into multi-agent social trajectory prediction, so that it can search for the high-performance NN tailored to the unique mobility patterns of interactive users instead of individuals?*

In Section 1.3.3, we outline our specific contributions in detail to address the aforementioned research questions.

### 1.2.4 Communication- and Computation-Adaptive Multi-Agent Trajectory Prediction Gap in Distributed Mobility Networks

In Section 1.2.1, Section 1.2.2, and Section 1.2.3, we focus on centralized trajectory prediction (including isolated-agent and collaborative-agent scenarios), where all users' datasets are collected, processed, and trained through a central server. However, such a modeling approach raises concerns about user privacy and data security, as well as communication network limitations, especially due to uploading large, private datasets to a centralized server. Therefore, in this section, we aim to broaden the scope of centralized multi-agent trajectory prediction, as outlined in Section 1.2.3, to include distributed collaborative trajectory prediction. This extension enables us to enhance the trajectory prediction process by simultaneously increasing privacy and scalability in computations and communication.

To address centralized ML issues, the research community has recently introduced Federated Learning (FL) as a distributed ML framework [57]. In this section, we aim to bring the FL paradigm to the field of mobility prediction. In the context of FL, each participant trains a local model using their own private dataset, and then sends the trained model's weights to a central server for aggregation into a global model. This global model is then sent back to the participants for further training to reduce generalization errors, with the entire process constituting a single *training round*. The retraining and aggregation steps are repeated for multiple rounds until convergence is achieved. By ensuring that location data remains private and localized on the user device, FL-based TP offers an intuitive solution to concerns regarding data privacy. Furthermore, the scalability of training large-scale networks with extensive mobility data is greatly enhanced by FL's ability to distribute and process data.

While FL has been explored in various fields, its application in the field of mobility prediction is relatively novel. The few existing FL works have achieved success in trajectory prediction, yet they suffer from several issues [23, 54, 87, 90]. Primarily, neural architecture inflexibility arises when all clients share the same NN to execute a task, which might not fit the features of each user mobility dataset and network performance conditions. The NN inflexibility in FL is even more severe than in centralized TP models due to the requirement of transmitting distributed local models over dynamic wireless networks. For example, a model might be too large for some clients with reduced computation and communication resources, and too small for others with complex datasets. Therefore, in this section, we aim to develop an advanced multi-objective NAS optimization technique that considers prediction performance and communication and computational costs together for designing a NN architecture tailored to the user's data features, computational resource capacity, and the wireless network's available throughput in an FL setting.

The other significant challenge with existing FL methods is the handling of ineligible training clients. In classical FL, the aggregation server must collect local models from all participating clients to compute a global model, resulting in computation delays due to local model training and communication delays due to model transmission. This method's limitation is that even if a single participating client is ineligible, meaning it has reduced computational or communication resources (i.e., a "straggler"), the whole federated global model training process is slowed down. Furthermore, the participation of ineligible low-quality data clients in federated training reduces overall global accuracy and increases the system's computational resource consumption. Hence,

in this section, we aim to develop data estimation techniques to enable the central server to estimate the local user dataset's quality without seeing their raw data. This way, the server can elect trustworthy and eligible users to participate in FL training rounds, reducing computational resource usage and communication overhead.

In the direction of the above limitations, we aim to address the following research questions.

- *RQ 4.1: How can we develop an approach to estimate the quality of local user datasets in an FL setting, which can be reported to the central server without the server having access to the raw data, in order to provide the server with a general understanding of the local users' potentials?*
- *RQ 4.2: How can we minimize computational resource usage and communication overhead in FL by selecting only a subset of eligible local users to participate in each federated training round?*
- *RQ 4.3: How can we design a multi-objective NAS algorithm that designs a high-performance NN simultaneously optimizing accuracy, training time, and transmission time, utilizing federated participants' datasets and resources and the varying network throughput, to achieve an optimal trade-off among these factors?*

In Section 1.3.4, we outline our specific contributions in detail to address the aforementioned research questions.

### 1.2.5 Strategically Robust Multi-Agent Trajectory Prediction Gap in Non-Cooperative Mobility Networks

In Section 1.2.4, we focus on distributed ML through the FL framework for private, network-adaptive, and computationally-efficient collaborative trajectory prediction. However, from another perspective, distinct from the ones presented in Section 1.2.4, an important challenge associated with centralized and decentralized social-aware trajectory predictors, is their difficulty in effectively handling non-cooperative social behaviors. Non-cooperative social interactions refer to situations where the agents do not actively collaborate or coordinate their actions toward a common goal. Instead, each agent acts independently, pursuing its own objectives without any shared strategy or cooperation with other agents. Therefore, each mobile user acts autonomously, making decisions solely based on their personal trajectory data and preferences. In Section 1.2.3 and Section 1.2.4, we focus on cooperative collaboration where the interacting mobile agents share their NN architectures, which might be against their personal desires. Such an approach in large-scale networks can end up in a reduced average prediction accuracy. Therefore, in this section, we aim to develop strategical non-cooperative social-aware trajectory predictors.

While centralized TP models are common in the literature, the decision-making power is taken away from individuals and given to a centralized unit to make optimal cooperative decisions by dedicating a single NN model to multiple input users of the multi-agent social-aware predictor. The problem is that not only the NN is heuristically designed but also a single model is shared among heterogeneous users. On the other hand, the challenge with the existing decentralized TP models, such as FL, is that aggregating through the common method of averaging locally trained models can limit the ability to personalize NN architectures. Although local models could get

personalized by individually tuning their neurons' weights, they must have identical architectures (identical number of layers, sequence of layers, and number of neurons) to enable matrix summation. This limitation hinders the ability of users to optimize their local NN architectures according to their specific data characteristics and movement strategies, which is crucial for accurate trajectory prediction.

Conducted surveys found that classical AI and game-theoretic approaches have the potential for modeling human behaviors in dynamic multi-agent systems. Therefore, in this section, we aim to expand the scope of centralized and decentralized cooperative multi-agent trajectory prediction to encompass non-cooperative multi-agent trajectory prediction within the context of Game Theory (GT). This allows us to enhance the trajectory prediction process by incorporating individual agents' distinct desires in multi-modal scenarios.

Currently, few research works conducted GT in the trajectory prediction field. However, these models have multiple limitations [53, 27, 5]. Firstly, the existing works are mainly designed for sequential games and may not be suitable for modeling simultaneous games. In TP problems, individuals may take actions simultaneously, making it necessary to use non-cooperative simultaneous games to more accurately model their impulsive behaviors.

Moreover, existing GT-based TP approaches do not personalize the NN architectures of non-cooperative users but instead, use *implicit layers* to learn the underlying relationships between input and output data to determine the best response of each agent in a Nash Equilibrium of the game. The problem with these approaches is that they do not take into account the unique characteristics of individual non-cooperative users when designing NN architectures. By using implicit layers to learn the relationships between input and output data, the resulting models may not be optimized for the specific data patterns and behaviors of each user.

In this section, our focus is on developing methods that can effectively handle non-cooperative behaviors by employing multiple contesting NAS-enabled NN architectures in a simultaneous game. The objective of these networks is to accurately predict the joint trajectories of multiple adversarial users within a multi-agent setting.

In the direction of the above limitations, we aim to address the following research question.

- *RQ 5: How can we extend game-theoretic techniques to incorporate multiple contesting NAS, enabling the accurate capture of cooperative and non-cooperative user behaviors in collaborative multi-agent trajectory prediction?*

In Section 1.3.5, we outline our specific contributions in detail to address the aforementioned research question.

## 1.3 Thesis Contributions

As mentioned in Section 1.2, there is a pressing need to enhance the performance of current trajectory and mobility prediction methods across various aspects. Given the complexity of dynamic urban environments, accurately forecasting the trajectories of pedestrians and vehicles presents considerable challenges. Current trajectory prediction techniques often have limitations and may

not fully account for the unique characteristics of various types of mobility data, social interactions among multiple interfacing users, strategic behaviors among multiple competing users, and the distributed nature of mobility data from decentralized devices in urban environments. We propose several research contributions in Section 1.3.1 to Section 1.3.5 to solve the research problem indicated in Section 1.2.1 to Section 1.2.5. These contributions involve the development of advanced NNs and intelligent decision-making techniques that can account for the unique characteristics of urban mobility data, improve the prediction accuracy, speed up the learning convergence, offer scalable solutions, bring privacy, adapt to wireless network resources, while reducing computational and communication costs in isolated, social collaborative, centralized, distributed, cooperative, and non-cooperative mobile-network scenarios.

The relation between our proposed five trajectory predictors is illustrated in Section 1.4, Figure 1.2. We begin with an isolated-agent personalized trajectory prediction, where individual users' preferences are taken into account separately. As we progress, we expand our approach to incorporate scalable personalization in large-scale networks. Furthermore, we introduce the concept of social-aware collaborative multi-agent prediction, harnessing group intelligence to capture interactions among mobile users. Our initial social-aware predictor operates in a centralized manner, focusing on resolving computational management issues rather than considering privacy concerns for multiple users. Subsequently, we transit to a distributed social-aware trajectory prediction approach by developing a network-adaptive federated learning model. This model ensures the preservation of privacy for a multi-agent scenario while addressing network adaptivity. Lastly, we introduce the concept of non-cooperative trajectory prediction, employing game theory to capture interactions among multiple mobile users who may not necessarily have joint strategies aligned with other individuals' desires.

### 1.3.1 Reinforced Deep Learning for Personalized and Computationally-Light Trajectory Prediction and Proactive Mobility Management

To solve the research problems discussed in Section 1.2.1 and as the first contribution of this thesis, we propose a reinforcement learning-based NAS method that automates the design of the LSTM neural network in a more efficient manner, achieving a desirable balance between high accuracy and low algorithm complexity. This model is called RL-LSTM and serves as the cornerstone of our work. The advantage of utilizing RL-based NAS optimization is its cumulative nature, which enables the partial training of different NN architectures within each episode for a limited number of epochs. The sequential and cumulative RL algorithm enables effective exploration of a vast search space, identifying high-performance NN architectures while minimizing computational costs. This is in contrast to other non-cumulative optimization methods that necessitate training each NN architecture for the complete number of epochs at once. Thus, the RL technique offers a significant reduction in computational complexity compared to grid search and Auto-ML NAS models, increasing the speed of ML convergence while maintaining high levels of accuracy.

Building upon our RL-LSTM framework, we introduce two mobility management systems: RL-HEC (proactive handover management system) and RL-SM (proactive service migration system). These systems serve as examples of how our RL-LSTM technique significantly enhances the performance of wireless network applications.

In the subsequent paragraphs, we provide an overview of our contributions, while a more comprehensive discussion of our proposed models can be found in Chapter 5 of this thesis. Additionally, for further in-depth information, we refer readers to our journal paper [104].

**LSTM Trajectory Predictor Design through RL**

Generally, deep learning models and NNs have a large set of hyperparameters, including activation functions, neural layer types, numbers, and sequential orders, and dropout values and units. For instance, LSTMs consist of LSTM, dense, and dropout layers. With the exponential growth of possible hyperparameter combinations, selecting the optimal NN architecture becomes an NP-hard optimization problem. As stated, exhaustive grid search, naive heuristics, or random search algorithms are not feasible or optimal solutions and cannot solve the NAS optimization problem within a polynomial time. Instead, utilizing reinforcement learning can offer a faster convergence solution and enable solving the NAS problem in polynomial time, overcoming the limitations of traditional search methods.

RL is an unsupervised technique that enables agents to make sequential decisions by learning from interactions with an environment. The agent receives feedback in the form of rewards or penalties based on the actions taken, and through this iterative process, the agent learns to identify the optimal policy that maximizes the cumulative reward. By reducing the computational complexity of the algorithm while identifying the optimal solution, RL significantly improves the solution quality, computational complexity, and convergence of trajectory prediction models with respect to other NN search mechanisms.

RL is a well-suited solution when dealing with non-convex objective functions, prioritizing the attainment of satisfactory solutions rather than finding the global optimum. Furthermore, RL demonstrates remarkable effectiveness in addressing problems that involve accumulating rewards over multiple iterations during state transitions, resembling the nature of Markov Decision Process (MDP). The NAS problem can be effectively formalized using the MDP concept within RL, which allows us to treat each NN architecture as an agent that makes decisions to maximize cumulative rewards over multiple training iterations. The MDP state transition captures the progression of the NN architecture through different configurations during the search process, and the accumulation of partial training at each iteration leads to the final accumulated knowledge, which guides the architecture towards optimal performance and improved computational efficiency in NAS.

In response to RQ 1.1, we address the design of high-performance LSTM neural network architectures using a reinforcement learning approach. Our aim is to leverage RL to optimize the architecture of LSTM trajectory predictors based on features extracted from specific mobility data in a more efficient way than other NAS mechanisms.

To achieve this, we employ RL to iteratively search and evaluate different LSTM architecture configurations. During each RL itteration or episode, the agent selects an action (a NN architecture) based on its policy. Then, the prediction accuracy (in classification problems) or mean squared error (in regression problems) of the suggested architecture is used as a reward metric for the RL agent. Such a feedback serves as a measure of how well the LSTM model performs in predicting trajectories based on the given mobility data. Through successive RL iterations, the search

space for LSTM architectures gradually narrows down. This narrowing occurs as the RL algorithm explores different combinations of neural layers, neuron counts, and other relevant parameters, and evaluates their performance using the accuracy metric. The RL agent learns from these evaluations and adjusts its exploration strategy to converge towards the optimal LSTM architecture that maximizes prediction accuracy for the specific dataset.

By incorporating RL into the architecture design process, we can systematically explore and optimize LSTM configurations, tailoring them to the unique characteristics and patterns present in the mobility data. This approach enables us to develop high-performance LSTM trajectory predictors that are well-suited for accurate trajectory prediction in a given dataset.

**Reinforced Trajectory Prediction-Driven Proactive Handover Management and Service Migration**

As a user moves to different areas, the network must react to route and topology changes and adjust them to ensure uninterrupted service. Such handover process involves multiple signaling protocols between the mobile user, source/target base station, and core networks. In modern wireless networks with dense cell placement, mobility between cells is frequent, resulting in an increase in the number of handovers. More frequent handovers can increase delay, decrease throughput, and lead to signaling overhead. Since service migrations occur reactively after handover events, delays in handovers can result in service discontinuity. Thus, robust predictive models that can accurately forecast the future behavior and mobility patterns of mobile users can greatly enhance the performance of handovers in wireless networks. Predictive models can also enable the network to anticipate handover events and prepare in advance, leading to more seamless transitions and improved overall user experience.

In response to RQ 1.2, we propose to design a proactive handover mechanism, so called Reinforcement Learning-based Handover for Edge Computing (RL-HEC), by integrating our proposed relible and convergent RL-LSTM mobility predictor into the handover decision. With this design, the source, and target base station exchange messages before the mobile user switches the connection [59]. Our proposed anticipatory model can be used to predict the occurrences of ping-pong handovers and also service migration patterns. In this direction, we have developed an RL-based personalized LSTM NN trajectory predictor, which is integrated with a handover decision-making mechanism. Our proposed scheme seeks to avoid ping-pong handovers and maximize service continuity by considering the specific services being consumed by end-users and connection information. This approach helps to prevent link failures and service disruptions that may be caused by handovers.

Additionally, we introduce a service migration framework, so called Reinforcement Learning-based Service Migration (RL-SM), that leverages the RL-LSTM mobility prediction model. This framework is specifically designed for edge computing scenarios and improves the performance of delay-sensitive services in a MEC environment. Overall, our proposed models provide an effective solution for ensuring smooth and uninterrupted network operations in the face of increasing user mobility and service demands.

Our various simulation results demonstrate that our proposed solutions have the potential to greatly reduce ping-pong handover rates, with some cases seeing a decrease to nearly zero. Additionally, the proposed solutions show an improvement in network throughput, with measurements indicating a 1.5 times increase compared to state-of-the-art solutions. This improvement in performance is accompanied by a notable reduction in the number of migration attempts and failures, indicating greater reliability and efficiency of the network.

### 1.3.2 Clustered Transfer Learning for Large-Scale Trajectory Prediction

To solve the problems discussed in Section 1.2.2 related to scalability issues of NAS techniques in large-scale networks, we present the second contribution of this thesis. We propose Reinforcement Convolutional Transfer Learning (RC-TL), a CNN-based scalable trajectory prediction system that clusters users with similar trajectories, dedicates a single RL agent per cluster to design a high-performance CNN architecture, trains one model per cluster using the data of a small user subset, and transfers the NN model to the other users in the cluster employing Transfer Learning (TL) techniques. These proposed methods aim to enhance the scalability of the NAS-enabled trajectory predictor while not sacrificing too much the mobility prediction accuracy in dynamic urban environments.

In the subsequent paragraphs, we provide an overview of the steps required to build RC-TL, while a more comprehensive discussion of our proposed models can be found in Chapter 6 of this thesis. Additionally, for further in-depth information, we refer readers to our paper on RC-TL [21].

#### 1D-CNN Trajectory Predictor Design through RL

In response to RQ 2.1, we propose to adopt One-dimensional (1D) CNNs for the task of trajectory prediction. CNNs are deep learning models that excel at capturing spatial and hierarchical patterns in 2D data, while 1D CNNs specifically specialize in extracting features and patterns from 1D sequential data. Thus, 1D-CNNs could be another approach to modeling sequential mobility data. Compared to RNNs, which have a sequential nature and can suffer from slow training and the vanishing gradient problem, CNNs are more robust and can efficiently parallelize the training process. In addition, 1D-CNNs can be more effective in modeling sequential data than RNNs. This is because 1D-CNNs consider the entire sequence of input data as a single input and apply a fixed-size filter to extract local features, rather than considering windows of sequences like RNNs. This allows 1D-CNNs to capture long-term dependencies in the input data more effectively.

CNNs contain convolutional, max-pooling, flatten, dense, and dropout layers, each of these layers serving a unique purpose. Designing a high-performance CNN involves exploring a vast search space, which can dramatically increase in size and complexity. In this direction, we propose a highly-accurate CNN architecture designed through RL (RL-CNN) for trajectory prediction in large-scale networks. The parallelization within CNN makes it a faster convergent predictor, making it well-suited for large-scale networks and providing better fitting capabilities.

#### Similar-Trajectory User Clustering

In relation to RQ 2.2, we group users with similar trajectories into disjoint clusters using the Longest Common Sub-Sequence (LCSS) similarity measure technique between every pair of user trajectories in the dataset, which populates a symmetric distance (proximity) matrix representing the difference between trajectories. Unlike other distance measures that require trajectories to have the same length or alignment, LCSS can handle sequences of varying lengths. It identifies the longest subsequence that is common to both trajectories, regardless of differences in the number of points or the temporal alignment.

We then apply clustering algorithms that use distances between points, such as Birch, DBSCAN, K-Means, Mean-Shift, Ward, and Optics, to compare pairwise the distances between locations within trajectories and group the trajectories into disjoint clusters.

#### Computational and Communication Cost Management through Transfer Learning

In response to RQ 2.2 and given clusters of similar-trajectory users, we propose training only a subset of users within each cluster who have high-quality and periodic data to become representative users to manage the scalability and computational resource usage in large-scale networks. The remaining users within the cluster can stay silent during the training phase, and the pretrained models can be transferred to them using the transfer learning paradigm. TL is based on the idea that the knowledge gained while solving one problem can be used to improve the performance of a related problem, speeding up the process and saving computational resources. By employing TL, we can reduce the computational cost of training and achieve better performance in the mobility prediction task.

This approach can result in significant resource savings, with the number of representative users being determined balancing system accuracy with computational resource usage. The selection of the number of representative users plays a crucial role in balancing system accuracy and computational resource utilization in RL-CNN training. Increasing the number of representative users enhances the model's generalization for intra-cluster features, resulting in improved accuracy. However, it also introduces higher computational complexity. The determination of an appropriate compromise between accuracy and computational resources depends on the specific requirements of the application at hand. By considering the application's specific needs, the optimal number of representative users can be determined, achieving the desired balance between accuracy and computational complexity.

### 1.3.3 Intra-Cluster Collaborative Learning for Social Trajectory Prediction

To solve the problems discussed in Section 1.2.3 related to lack of an efficient NAS and high computational costs in the existing social-aware trajectory prediction methods, we present the third contribution of this thesis. We propose Intra-Cluster Reinforced Social Transformer (INTRAFORCE) trajectory prediction system as a high-performance and low-complex collaborative predictor. INTRAFORCE uses RL to build a Social-Transformer architecture that learns the social interaction within clusters of similar mobile users based on their intra-cluster mobility features.

In the subsequent subsections, we provide an overview of the steps required to build INTRAFORCE, while a more comprehensive discussion of our proposed models can be found in Chapter 7 of this thesis. Additionally, for further in-depth information, we refer readers to our paper on INTRAFORCE [20].

**Transformer Trajectory Predictor Design through RL**

In response to RQ 3.1, we propose an optimized and automated trajectory predictor based on transformer NNs using reinforcement learning. Transformers are a type of deep learning model that utilize self-attention mechanisms to capture complex patterns and dependencies in sequential data. While CNNs are known for their efficiency, particularly in tasks involving grid-like data such as image recognition, transformers have demonstrated exceptional performance in Natural Language Processing (NLP) and machine translation tasks, thanks to their self-attention mechanism that captures long-range dependencies and contextual relationships in sequential data. Considering the task of mobility prediction as a time-series prediction, transformers can be a more promising alternative to both RNNs and CNNs due to their ability to handle sequential data efficiently. Hence, our proposal revolves around utilizing transformers as the core architecture for our social trajectory predictor, as accurately capturing user interdependencies is of utmost importance.

Transformers are composed of Encoder and Decoder stacks with multi-head attention layers, add and norm layers, feed-forward layers, and dropout layers. Optimizing a high-performance transformer from a vast search space demands an efficient NAS mechanism. Therefor, we optimize and fine-tune our transformer-based social-aware trajectory predictor through our previously proposed RL technique.

**Intra-Cluster Social Interaction Extraction**

In response to RQ 3.2, we propose a multi-modal TP that takes into account the social interactions between users within a cluster (intra-cluster users) through a social-pooling layer, who are grouped based on their geographical proximity and similarity. The social TP is achieved through the utilization of a social-pooling layer. Clustering adjacent trajectories enables us to compute the social interactions only among users who have a significant impact on each other, rather than considering all users in the mobility scenario as existing methods do. By focusing on intra-cluster interactions, we can reduce the computational requirements for social TP models. This is because we only consider interactions between users in the same cluster, rather than interactions between all users in the system.

**Social-Transformer Trajectory Predictor Design through RL**

In response to RQ 3.3 and in order to create a high-performance multiple-input multiple-output social neural network, we incorporate our proposed RL technique from Section 1.2.1 into the intra-cluster users. Due to the highly similar mobility features among users within the same cluster, only one RL training per cluster of socially impactful users would be sufficient for personalizing the social TP model, ensuring efficient resource utilization without unnecessary wastage. Consequently,

in our proposed approach, a single user within each cluster is selected to train the reinforcement learning-based transformer (RL-TF) model, which is then shared and migrated to the other users within the same cluster. By leveraging this knowledge transfer, the architecture and learned insights are disseminated among the cluster mates. Subsequently, all the users within the cluster are combined using a social pooling mechanism to form a cohesive multi-agent trajectory predictor. This collective approach allows for the integration of diverse user behaviors and interactions within the cluster, leading to improve social trajectory prediction capabilities.

### 1.3.4 Network-Adaptive Federated Learning for Distributed Trajectory Prediction

To solve the problems discussed in Section 1.2.4 related to lack of multi-objective NAS and ineligible participants in distributed trajectory prediction, we present the fourth contribution of this thesis. We propose Network-adaptive Federated Learning for Reinforced Mobility Prediction (FedForce) system. FedForce is a distributed system that employs RL-based NAS to design a transformer NN architecture that jointly optimizes multiple objectives of prediction accuracy, training time, and transmission time based on the mobility dataset's unique features, the client's computing capacity, and the available network throughput.

Moreover, in classical FL, the aggregation server collects local models from all participating clients to compute a global model. However, this approach faces challenges in terms of training delays and accuracy reduction due to existence of participants with limited computational or communication resources or low-quality data. The FL paradigm prevents the central server from accessing raw local data, making it unable to distinguish eligible users from ineligible ones. To address this issue, FedForce introduces a local data estimation technique. This enables local devices to locally compute the quality and periodicity of their data and report this information to the server using a single value, allowing for more efficient and accurate participation in the FL process.

In the subsequent subsections, we provide an overview of the steps required to build FedForce, while a more comprehensive discussion of our proposed models can be found in Chapter 8 of this thesis. Additionally, for further in-depth information, we refer readers to our paper on FedForce [18].

#### Time- and Frequency-Domain Data Signal Processing for Data Quality Estimation and Eligible User Selection

Regarding to RQ 4.1 and RQ 4.2, to manage the computational resource costs while increasing the global accuracy and decreasing the latency created by stragglers, we propose selecting an eligible subset of users for federated training while the rest of the system users can stay silent. However, a critical question arises concerning the determination of eligible users who can effectively contribute as representative participants in a decentralized federated system. In contrast to centralized models, in FL the eligibility of participants becomes crucial as the inclusion of ineligible users can significantly impact prediction accuracy or introduce system slowdowns. Selecting representative users in a decentralized system is harder than in a centralized system because it requires accounting for various challenges associated with distributed data, diverse user characteristics,

and limited communication capabilities. In a decentralized setting, the data is spread across multiple local devices or nodes, making it difficult to access and aggregate. Additionally, each user may have unique characteristics and behaviors that need to be considered in the selection process. Furthermore, the limited communication between nodes adds complexity in coordinating the selection process and exchanging information. These challenges necessitate the development of sophisticated algorithms and strategies to ensure fair representation and effective collaboration in decentralized environments.

In response to RQ 4.1, we propose a data quality estimator metric called *regularity ratio* by processing the mobility data, which is a time-series signal, in both time and frequency domains. Afterward, we establish specific thresholds to filter out eligible users based on their data quality. The acceptable value for regularity ratios is determined empirically based on the characteristics of the available dataset. By combining information from both the time and frequency domains, we can estimate the quality and periodicity of user data with a reasonable degree of accuracy. This approach enables the identification of regular mobility patterns, which are easier to analyze and predict.

In response to RQ 4.2, we propose selecting representative or eligible users for federated training in distributed systems based on the highest regularity ratios while considering users who satisfy both ratios. By prioritizing users with the highest ratios and meeting the defined criteria for both ratios, we aim to ensure the inclusion of highly representative and reliable participants in the federated training process. Afterwrads, the global federated model trained by these participants can then be transferred to the silent users, significantly reducing computational resources and communication overheads by minimizing the need for communication signaling. By keeping a portion of ineligible users silent, not only are computational and communication costs reduced, but the risk of stragglers causing interference and slowing down the entire FL system is also minimized. Carefully choosing the number of local participants is essential to achieving an optimal equilibrium between the overall system accuracy and the efficient utilization of computational resources. Similarly, selecting the appropriate number of pre-trained model migrations to all users is vital in maintaining the right balance between the overall system accuracy and minimizing communication overheads.

In the following paragraphs, we present the detailed methodology of our proposed regularity ratio metric, which serves as a data quality estimator in both the time and frequency domains.

1. Time Domain: Based on extensive experiments through training and testing heterogeneous data users, we have inferred that users who produce more data samples while visiting relatively fewer locations display more regular mobility patterns, leading to better TP accuracy. Thus, the *regularity ratio* in *time domain* is defined as the ratio between the number of total data samples and the number of unique visited locations, providing a score to estimate the quality and periodicity of the data associated with each user. Users with relatively higher regularity ratio have visited a limited set of distinct locations multiple times, making it easier for a NN to infer periodic behavior compared to rest of users with extremely low or high regularity ratios.

2. Frequency Domain: Our observations also indicate that converting the timeseries signal to *frequency domain* can provide valuable insights for predicting user mobility patterns. Specifically, a high power spectral density with a dominant frequency that is sufficiently high, coupled with a high Signal-to-Noise Ratio (SNR), suggests the presence of a strong and well-defined oscillation at a high frequency, with relatively low levels of noise. This results in user data that is easier to analyze and predict, as the oscillations are distinct and identifiable. In contrast, a low dominant frequency indicates a longer or more irregular pattern of oscillations, while a very high dominant frequency suggests a more complex and less predictable pattern. We define the regularity ratio in the frequency domain as the ratio between the SNR and the dominant frequency of the user's time-series data signal. Users with relatively high dominant frequencies and high signal SNR typically exhibit strong oscillations in their sequential data and frequently visit multiple locations, making it easier for a NN to detect periodic behavior. In contrast, users with extremely low or high dominant frequencies may have visited a wide range of locations or not moved significantly, resulting in non-periodic behavior.

**Network-Aware Federated Learning Design through Multi-Objective Reinforcement Learning**

Many existing distributed TP works utilize heuristically-designed NN architectures. However, only a few existing works that apply NAS on NN optimization consider only the predictors performance without taking clients' computational and communication resource limitations into account. On the other hand, a few works consider the utilization of computational resources but do not take into account the transmission of model size from local devices to the central server over the wireless network or the accuracy of the predictor. To address this research gap and in response to RQ 4.3, we propose a multi-objective RL-based NAS for federated TP. We convert the multi-objective optimization problem into a single-objective optimization problem by defining a cost function based on the linear combination of three parameters of performance, computational cost, and communication cost.

In Section 1.3.4, the objective or cost function of the RL agent, in the form of a reward function, is to search for a transformer architecture that satisfies all three parameters of accuracy, train time, and transition time. In contrast to our contributions in Section 1.3.1 to Section 1.3.3, the reward function of the RL agent aimed to suggest neural network architectures that only ensure prediction accuracy, without considering training time or model size of the suggested NNs based on the communication and computational limitations. The transmission time of a model with a specific size is significantly influenced by the available network throughput. In FedForce, the RL component considers varying bandwidths and adjusts the neural network sizes according to the available throughput. This network-adaptive approach ensures efficient utilization of the network resources, enabling FedForce to adapt to different network conditions.

In multi-objective optimization problems, designing a neural model with multiple objectives, such as accuracy, model size, and latency, introduces increased complexity compared to solely optimizing for accuracy. This complexity highlights the advantages of using reinforcement learning in comparison to AutoML in addressing such nonlinear, interconnected NAS problems. Therefore, our RL-based proposed technique, tailored for multi-objective optimization problems, provides an even more efficient and effective solution in NAS context.

### 1.3.5 Inter-Cluster Game-Theoretic Learning for Non-Cooperative Trajectory Prediction

To address the challenge outlined in Section 1.2.5 concerning the lack of robustness in developing models capable of accommodating both cooperative and non-cooperative user preferences, we introduce the fifth and final contribution of this thesis. We propose Game-Theoretic Trajectory Prediction through Distributed Reinforcement Learning (GTP-Force) system. GTP-Force is designed to construct a highly accurate trajectory predictor that captures the social interactions among multiple competing mobile users. GTP-Force employs RL to develop a high-performance transformer NN architecture based on *intra-cluster* users, known as cooperative users with similar data and preferences. Next, our approach utilizes non-cooperative game theory to determine the optimal combination of *inter-cluster* NN architectures in a dynamic multi-agent environment.

To minimize computational expenses, we assumed cooperative behavior among intra-cluster users since they have similar trajectories, and instead model a game among inter-cluster users who are definite competitors. In addition to effectively managing resource usage, this approach enables us to model both cooperative and non-cooperative users within a multi-agent scenario. By considering the diverse behaviors and preferences of users, we can develop a comprehensive framework that captures the dynamics of interactions among agents, accommodating cooperative behavior as well as individualistic and non-cooperative tendencies.

In the subsequent subsections, we provide an overview of the steps required to build GTP-Force, while a more comprehensive discussion of our proposed models can be found in Chapter 9 of this thesis. Additionally, for further in-depth information, we refer readers to our paper on GTP-Force [19].

#### Intra-Cluster Cooperative Predictor Design through RL

Existing social-aware TP models assume cooperative behavior from all users in a multi-agent environment and use a single NN model to predict trajectories of all users. However, in reality, human agents tend to optimize their personal goals instead of joint strategies, leading to inaccuracies in joint TP, particularly for *inter-cluster* users with differing trajectory features. Conversely, *intra-cluster* users, who exhibit similar mobility characteristics, can take advantage of their shared traits by utilizing a common neural network model. This approach improves prediction accuracy and addresses the challenges posed by heterogeneous user behaviors in multi-agent TP.

To address the *cooperative* part of RQ 5, our proposed approach involves applying a single reinforcement learning technique to each cluster. This allows us to design multiple high-performance transformer NNs tailored to the unique mobility patterns of inter-cluster users. By doing so, we aim to enhance the predictive capabilities of the model and accommodate the specific requirements of different user groups within the multi-agent system.

#### Inter-Cluster Non-Cooperative Multi-Modal Game Design

To address the *non-cooperative* part of RQ 5, we present a novel approach that incorporates a non-cooperative game model for inter-cluster users. This model takes into account the individual goals of each user while maintaining overall system accuracy. Non-cooperative agents strategically

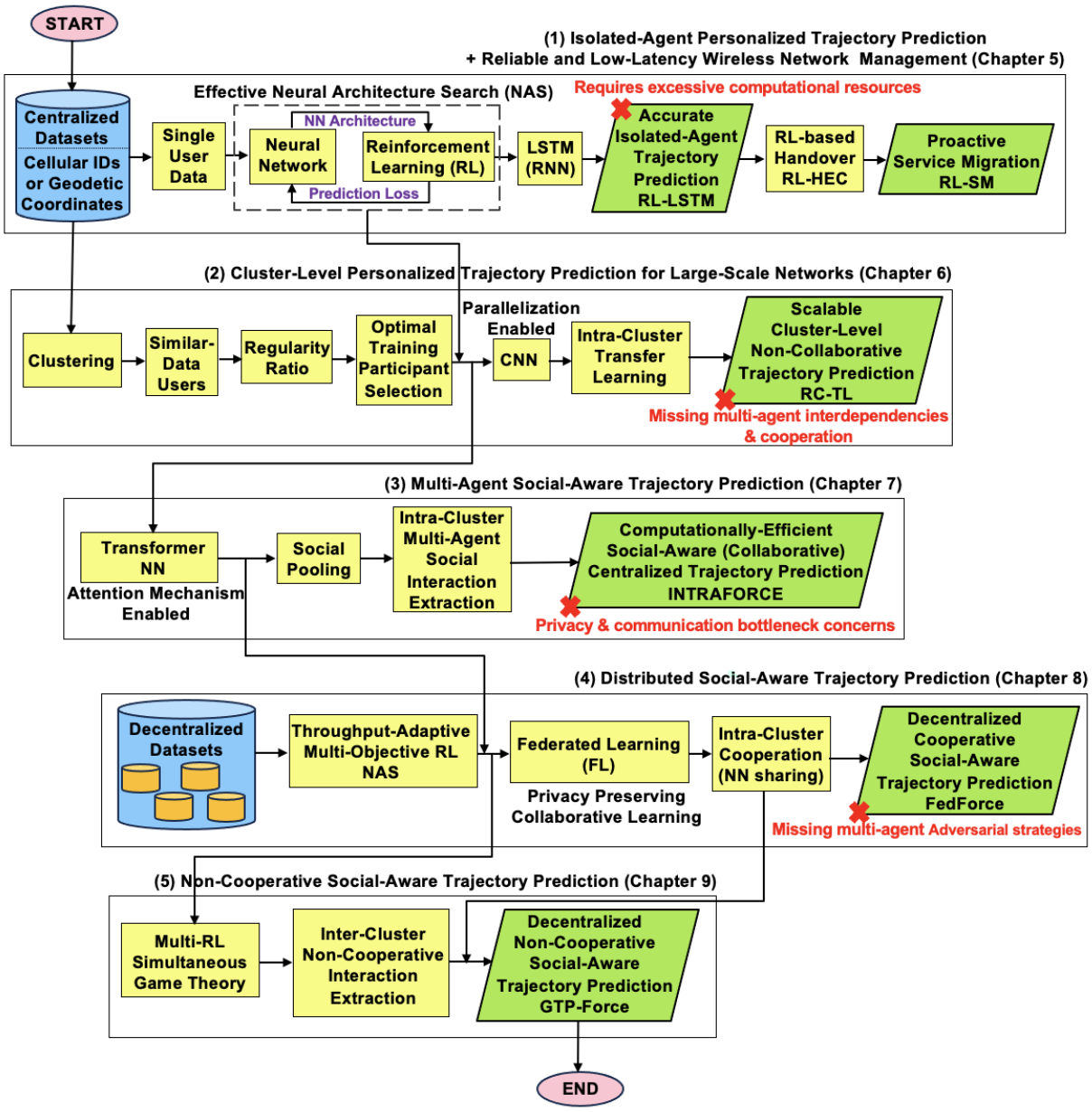



FIGURE 1.2: Thesis contribution workflow diagram.

select a combination of competing neural networks that optimize individual players' decisions, while simultaneously enhancing the overall system strategy.

Hence, the role of RL goes beyond designing the highest-performance neural network for each player or cluster. Instead, in our approach, each player, represented from a distinct cluster, contributes a list of high-performance neural networks, and through a simultaneous game, a combination of multi-agent RL determines the optimal configuration that benefits the entire competitive system.

In our distributed GTP-Force system, game players upload only their RL-trained weights and not data, ensuring efficient communication. Each player in the game possesses a collection of potential high-performance transformer architectures. To train a multi-agent trajectory prediction model and evaluate the impact of each player's decision on others, a social pool layer is incorporated in the central server. This facilitates the formation of a game's payoff matrix.

Finally, by analyzing the Nash equilibrium, GTP-Force identifies the most rational combination of NNs for the multi-agent environment. This approach enables us to strike a balance between individual user objectives (and their intra-cluster mates) and the collective accuracy of the system (inter-cluster players), ultimately enhancing the TP model's effectiveness.

## 1.4 Thesis Outline

Figure 1.2 presents a complete diagram of our five contributions (from Section 1.3.1 to Section 1.3.5) including five high-performance trajectory predictors. This diagram offers a comprehensive visual representation of our proposed mobility prediction and management systems which are elaborated in Chapter 5 to Chapter 9. This diagram comprises 5 main blocks, each representing one of our proposed contributions. The first block involves personalized isolated-agent trajectory predictors for individual users, leveraging LSTM and RL optimization techniques for accurate and reliable predictions, efficiently solving the NP-hard problem of NAS. In the second block, user data is clustered, and cluster-level personalization is applied, ensuring scalability with CNNs for parallelized training and faster convergence. The third block addresses isolated-agent prediction challenges by capturing interdependencies among neighbor-trajectory users after clustering in a computationally-efficient manner, while also optimizing transformer architectures for time-series mobility data through RL. In the fourth block, we transition to distributed ML using federated learning, with RL optimizing multiple objectives of accuracy, communication overhead, and computational resource consumption. Lastly, the fifth block demonstrates our work in resolving non-cooperative strategies among inter-cluster users with distinct desires using multiple contesting RL agents.

The rest of the thesis is as follows.

Chapter 2 describes in detail the theoretical background of mobility prediction and mobility management models and techniques.

Chapter 3 provides an overview of the related works of the domains that are relevant to our research, including state-of-the-art isolated, multi-agent, centralized, distributed, cooperative, and non-cooperative trajectory prediction models. The chapter discusses the relevant literature and

research that have been conducted in these areas and provides a comprehensive review of the limitations of existing approaches.

Chapter 4 provides a comprehensive understanding of the user mobility context by defining mobility and trajectory prediction, describing the mobility scenario, and presenting the details of the datasets, and data pre-processing and feature extraction techniques used to train various NNs in this study. The chapter also presents a detailed overview of our experimental setup and the evaluation metrics used to assess the performance of our proposed trajectory prediction models.

Chapter 5 presents the RL-LSTM trajectory predictor used to develop proactive handover management system (RL-HEC) and proactive service migration system (RL-SM) in detail with their framework architectures and operations.

Chapter 6 provides a detailed overview of the architecture and performance of the RC-TL trajectory predictor, including its clustering module, RL agent, and CNN-based predictor, and presents experimental results demonstrating its high prediction accuracy and reduced computational resources.

Chapter 7 provides a detailed description of the INTRAFORCE, which builds a trajectory predictor that learns the social interaction within clusters of similar mobile users using RL. The chapter covers the system's architecture, including its social-transformer architecture and intra-cluster user mobility features, and presents comprehensive experimental results demonstrating its superior accuracy and computational resource management compared to state-of-the-art models.

Chapter 8 presents the FedForce system, which employs RL to design a transformer NN architecture that jointly optimizes prediction accuracy, training time, and transmission time. The chapter provides a detailed description of the system's architecture, including the RL agent, the FL framework, and the joint optimization approach, and presents experimental results demonstrating its superior performance in terms of accuracy, training time, and communication overhead compared to state-of-the-art predictors.

Chapter 9 provides a detailed description of the GTP-Force system, which constructs a highly accurate trajectory predictor that captures the social interactions among inter-cluster competing mobile users using multi-agent reinforcement learning modelled through a simultanious non-cooperative game. The chapter covers the system's architecture, including the multiple decentralized competitive RL agents and the game-theoretic approach. The chapter then presents experimental results demonstrating its superior accuracy and computational resource management compared to state-of-the-art models.

Finally, Chapter 10 summarizes and concludes the contributions of this thesis and provides an overview of future research directions that could build upon the work presented in this thesis.

# Chapter 2

# Theoretical Background

## 2.1 Chapter Introduction

In this chapter, we investigate the theoretical background that underlies mobility prediction, mobility management, predictive ML models, and intelligent decision-making techniques used in this thesis. We explore the principal ideas, methodologies, and techniques that form the foundation of these fields. Our aim is to provide the reader with a deeper understanding of the key components, which will serve as a basis for the subsequent chapters.

In Section 2.2, we provide the theoretical foundations and mathematical principles underlying mobility prediction. In Section 2.3, we focus on mobility management, specifically Hanover management and service migration techniques. Moving forward, in Section 2.4, we delve into supervised machine learning predictors, while in Section 2.5, we discuss unsupervised clustering methods. Furthermore, Section 2.6 explores reinforcement learning, Section 2.7 covers transfer learning, Section 2.8 addresses collaborative learning, Section 2.9 explores federated learning, and Section 2.10 investigates game theory principals. These comprehensive discussions lay the groundwork for the development of our mobility prediction contributions in this thesis.

## 2.2 Mobility Prediction Definition

Mobility prediction involves anticipating the future movements or locations of mobile devices or users based on historical data, patterns, or trends as they navigate through different locations. It is a critical component in many areas, including wireless communication networks, urban planning, and transportation systems.

While mobility prediction can be viewed as location prediction, particularly when predicting a mobile user's next (1-hop) location, it is not limited to this definition. Mobility prediction can also involve predicting more detailed characteristics of the movement, such as speed, direction, and acceleration. When predicting a sequence of multiple locations, we often refer to it as trajectory prediction. This involves predicting the entire path that a user or device will follow over a certain period, not just the next immediate location. A trajectory is defined as the path followed by a mobile user, either a human, a vehicle, or a robot, as it moves through space or time. The trajectory

of a mobile user is typically derived from their mobility or historical sequence of visited locations, which can include GPS coordinates or wireless base stations [65].

However, mobility prediction is not only about location or trajectory prediction. It may also involve predicting behavioral patterns related to mobility, such as the time a user spends in a particular location (dwell time), regularity in movement patterns (e.g., daily commute to work), or predicting large-scale population movements (e.g., during a concert or a football game) [65]. It can also encompass the prediction of handovers in a cellular network context, where the goal is to predict when a mobile device will switch from one cell tower to another.

In this thesis, when we mention mobility prediction, our primary focus is on trajectory prediction. We extend the context of trajectory prediction beyond individual users to encompass group users. Moreover, we shift the focus from isolated individual trajectories to predicting the collective behavior of multiple interactive users through social prediction. Additionally, we transition from centralized prediction approaches to decentralized ones. Lastly, we explore the challenges and complexities of non-cooperative prediction involving multiple users, moving beyond the domain of cooperative prediction stratgies.

## 2.3 Mobility Management Definition

Mobility management refers to the set of techniques, protocols, and strategies employed to optimize and manage the mobility of users in a wireless network. It encompasses various aspects such as handover management and service migration. Handover management ensures continuous connectivity as mobile devices transition between different wireless network cell boundaries, enabling a seamless user experience. Service migration involves dynamically transferring ongoing services to other network nodes after handover management, facilitating uninterrupted service delivery [65].

By leveraging accurate mobility prediction, mobility management aims to achieve efficient resource allocation, minimize service disruptions, and enhance the overall user experience. It allows network operators to proactively adapt their infrastructure, allocate resources effectively, and optimize network performance based on anticipated user mobility patterns. Effective mobility management contributes to seamless connectivity, improved service quality, and enhanced user satisfaction in wireless communication systems. Therefore, the primary objective of this thesis is to develop robust mobility and trajectory predictors that effectively enhance the performance of mobility management services in modern wireless networks. By focusing on this direction, we aim to create predictive models that can accurately anticipate and adapt to users' mobility patterns, enabling more efficient and reliable network management.

### 2.3.1 Handover Management Definition

Handover management refers to the process of transferring an ongoing communication session from one base station to another as the user moves through a wireless network. The handover decision is typically based on the received signal strength, and various algorithms can be used to determine when and to which base station the handover should occur. One common approach to handover management is to take into account the current signal strength and the historical signal

strength of the current and neighboring base stations [65]. The handover decision is made based on an empirical threshold, which is defined as the minimum difference in signal strength between the current base station and the neighboring base station before a handover is initiated.

Let $RSS_{n,b}$ denote the received signal strength (RSS) at UE $n$ from base station $b$, and let $T_h$ denote the hysteresis threshold. The hysteresis-based handover decision can be made using the following equation:

$$H_{n,b} = \begin{cases} 1 & \text{if } RSS_{n,b} - \max\limits_{b' \neq b} RSS_{n,b'} > T_h \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $H_{n,b}$ is a binary variable that indicates whether or not UE $n$ should handover to base station $b$.

Another approach to handover management is to use a prediction-based algorithm, which utilizes a mobility prediction model to predict the future location and signal strength of the UE [65]. The handover decision is made based on the predicted signal strength of the neighboring base stations and the quality of the prediction model. Let $P_{n,b}$ denote the predicted signal strength of UE $n$ from base station $b$, and let $Q_n$ denote the quality of the mobility prediction model associated with UE $n$. The prediction-based handover decision can be made using the following equation:

$$P'_{n,b} = \begin{cases} P_{n,b} & \text{if } x_n = 1 \\ \max\limits_{b' \neq b} RSS_{n,b'} & \text{otherwise} \end{cases} \tag{2.2}$$

where $P'_{n,b}$ is the predicted signal strength of UE $n$ from base station $b$ after taking into account the presence or absence of the mobility prediction model. If the model is present ($x_n = 1$), the predicted signal strength $P_{n,b}$ is used. Otherwise, the maximum RSS from the neighboring base stations is used as the prediction. The handover decision is then made based on the predicted signal strength $P'_{n,b}$, and the quality of the prediction model $Q_n$.

### 2.3.2 Service Migration Definition

To minimize end-to-end latency in low-latency wireless applications, edge services should always be close to their user to some extent. However, simply transferring the service to the closest edge server ignores the dynamic nature of edge-enabled networks, such as how other users may reserve the available resources of the servers at the moment of the migration request. To address this issue, we can use a resource-aware service migration optimization algorithm that takes into account the dynamicity of the network and the available resources of nearby servers [65].

The resource-aware service migration optimization algorithm can be formulated as follows:

- Determine the next base station $B_n^{t+1}$ that user $n$ will connect to based on the mobility prediction scheme.
- Use the learning model $M_{b,k}$ associated with edge data center $E_b$ to determine the optimal base station $k$ for hosting the service $S_{n,b}$, taking into account the resources available at nearby edge data centers.

- If the optimal base station $k$ is different from the current base station $b$, initiate a service migration from base station $b$ to base station $k$.
- Monitor the resources available at nearby edge data centers to ensure that the service migration does not disrupt the services being consumed by other users.

We can use the following equations to implement the above algorithm: To determine the next base station $B_n^{t+1}$ that user $n$ will connect to based on the mobility prediction scheme:

$$B_n^{t+1} = f(B_n^t, B_n^{t-1}, \ldots, B_n^{t-m}) \tag{2.3}$$

, where $f$ is a function that takes into account the user's $m$ previous locations and predicts the next base station. To determine the optimal base station $k$ for hosting the service $S_{n,b}$:

$$k = \text{argmin} l \in 1, 2, \ldots, B(D_{b,l} + R_l + W_l) \tag{2.4}$$

, where $D_{b,l}$ is the distance between base station $b$ and base station $l$, $R_l$ is the remaining resources at edge data center $E_l$, and $W_l$ is the weighted average of the resources currently being used by the services hosted at edge data center $E_l$. To initiate a service migration from base station $b$ to base station $k$:

$$S_{n,k} = S_{n,b} \tag{2.5}$$

, where $S_{n,k}$ denotes the service being consumed by user $n$ at base station $k$. To monitor the resources available at nearby edge data centers: If the remaining resources $R_l$ at an edge data center $E_l$ fall below a certain threshold, the migration algorithm can avoid overloading the server by temporarily blocking new migration requests to that server until the resources are replenished: if $R_l <$ threshold, temporarily block new migration requests to $E_l$.

## 2.4 Supervised Machine Learning

In this section, we explore supervised ML techniques used in this thesis to develop resilient trajectory predictors. Supervised learning involves training the models using labeled data, allowing them to learn patterns and make predictions based on known trajectories. By leveraging the power of supervised ML, we aim to improve the accuracy and effectiveness of mobility prediction, enabling better understanding and anticipation of user movements in various contexts and scenarios.

In trajectory prediction, labeled data refers to the mobility dataset where each sample is associated with a corresponding ground truth label. For historical time and location data for individuals, the labeled data could include a sequence of past time and location observations as the input, and the target label would be the actual or observed location at a future time. This labeled data is used to train a trajectory prediction model to learn the underlying patterns and relationships between time, location, and movements.

In this thesis, we develop LSTM, CNN, and attention-based transformer NNs for trajectory prediction. In the following sections, we will explain the performance of these predictors.

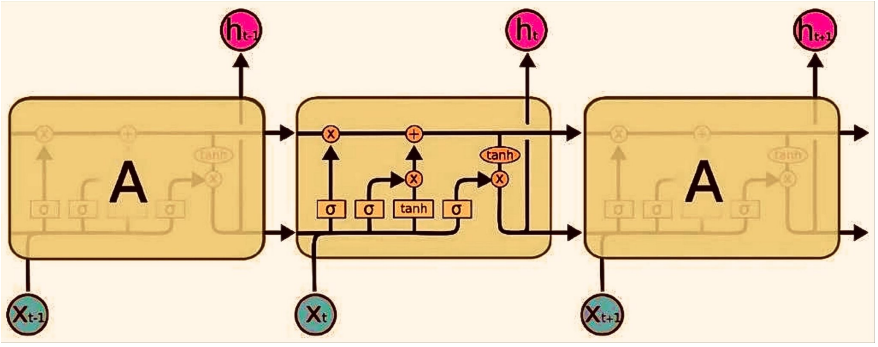



FIGURE 2.1: Stacked LSTM architecture [104] ©2022 IEEE.

### 2.4.1 LSTM Neural Networks

LSTM is a type of RNN commonly used for sequence data processing, such as time series or NLP tasks. Unlike traditional RNNs, LSTM networks are designed to capture and remember long-term dependencies in sequential data. They achieve this by incorporating memory cells and various gating mechanisms that control the flow of information through the network. This enables LSTMs to effectively model and retain important information over longer time intervals [65].

The LSTM neural architecture consists of a chain of memory cells, as shown in Figure 2.1. These memory cells are responsible for transferring information at different time steps, and each cell has three gates that control the flow of information: the input gate, the forget gate, and the output gate. Each gate contains a sigmoid layer that decides how much information passes through the memory blocks. The presence of three sigmoid $\sigma$ gates can be observed within each memory cell depicted in Figure 2.1. The input gate controls which part of the input will be utilized to update the cell state, while the forget gate controls which part of the old cell state will be discarded. Additionally, the output gate controls which part of the cell state will be exposed as the output. Together, these gates allow the LSTM to selectively remember or forget information from previous time steps, making it particularly effective for time-series data analysis [65].

The LSTM NN's performance can be defined based on a set of equations that calculate the output of a recurrent neural network by taking into account the input, previous output, and memory state. The equations are given as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{2.6}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.7}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{2.8}$$

$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \tag{2.9}$$

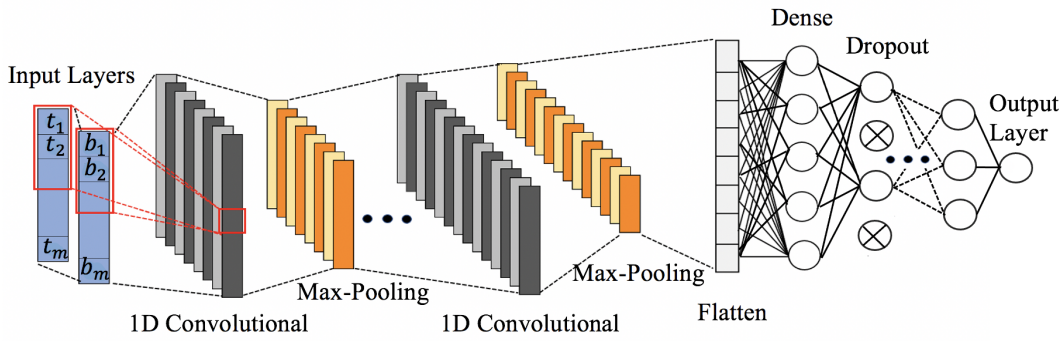



FIGURE 2.2: Structure of the generic 1D-CNN built and trained by RC-TL [21] ©2022 IEEE.

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.10}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.11}$$

where $i_t, f_t, o_t, g_t, c_t$, and $h_t$ are the input, forget, output, cell, memory, and hidden states of the LSTM cell at time step $t$, respectively. $x_t$ is the input at time step $t$, $h_{t-1}$ is the hidden state at the previous time step, and $W$ and $b$ are weight and bias parameters, respectively. As stated, the symbol $\sigma$ represents the sigmoid activation function, while $\odot$ represents element-wise multiplication, and tanh represents the hyperbolic tangent activation function. The equations describe the flow of information within the network, including how inputs are processed, how the memory state is updated, and how the output is calculated. The equations also include activation functions, such as sigmoid and hyperbolic tangent, which are commonly used in neural networks to introduce non-linearity.

### 2.4.2 1D-CNN Neural Networks

CNN is a deep learning architecture designed for processing and analyzing structured grid-like data, such as images, by applying convolutional filters to capture local patterns and hierarchically extract meaningful features. CNNs are inspired by the organization of the visual cortex in animals and use a hierarchical structure to learn and extract features from the input data. Unlike fully connected neural networks, which have connections between all input and output neurons, CNNs use sparse connections and parameter sharing to efficiently learn the relevant features in the input signal. In a CNN, the input is typically a multidimensional array or tensor, such as a color image represented as a three dimensional (3D) array of pixel values [65].

One dimensional CNN is a variant of the convolutional neural network architecture specifically designed for processing sequential or time-series data, where the input data has a linear structure. It applies one-dimensional filters over the input sequence to capture local patterns and extract relevant features, enabling effective analysis and prediction tasks in domains like time

series, NLP, and audio processing. The 1D-CNN architecture is similar to the standard CNN. Figure 2.2 shows the generic 1D-CNN developed in this thesis.

A CNN is composed of *layers*, each with a different purpose, e.g., convolutional, max-pooling, flatten, dense, and dropout layers. A convolutional layer applies the convolution operator between the input data $x$ and a set of *kernels* $w_k \in \mathbb{R}^s$, producing a set of *feature maps* $y_i = x * w_k$, where $s$ is a fixed kernel size. The goal of the convolutional layers is to detect the presence of spatio-temporal patterns in the input data. The size and the number of the different used kernels are decided at design time, whereas the numerical values of the kernels $w_k$ are computed during the model training process. The max-pooling layer reduces a feature map's dimension by partitioning it into same-size neighborhoods (strides) and generating a smaller feature map replacing each neighborhood of the original feature map with the maximum value of each neighborhood. A flattening layer converts a feature map to a one-dimensional array of features. A dense layer comprises a set of perceptrons that are fully connected to the previous and following layers. Finally, a dropout layer is used to reduce over-fitting by setting each element of an input array to 0 with a fixed probability called *dropout ratio* during the training [65].

The formula for a convolutional layer in a CNN is given by:

$$y_{i,j} = \sigma\left(\sum_{k=1}^{K}\sum_{l=1}^{L} w_{k,l}\, x_{i+k-1,j+l-1} + b\right), \tag{2.12}$$

where $y_{i,j}$ is the output activation at position $(i, j)$ in the feature map, $x_{i+k-1,j+l-1}$ is the input activation at position $(i + k - 1, j + l - 1)$ in the receptive field, $w_{k,l}$ are the learnable weights of the filter, $b$ is the bias term, and $\sigma$ is the activation function.

Consequently, the formula for a convolutional layer in a 1D-CNN is similar, but the input and filter tensors have only one spatial dimension:

$$y_i = \sigma\left(\sum_{k=1}^{K} w_k\, x_{i+k-1} + b\right), \tag{2.13}$$

where $y_i$ is the output activation at position $i$ in the feature map, $x_{i+k-1}$ is the input activation at position $i + k - 1$ in the receptive field, $w_k$ are the learnable weights of the filter, $b$ is the bias term, and $\sigma$ is the activation function.

### 2.4.3 Transformer Neural Networks

Transformers were developed after the concept of attention-based neural networks. Attention mechanisms were initially introduced as an enhancement to RNNs and LSTMs to address the limitation of capturing long-range dependencies in sequential data. Early attention models, such as the Bahdanau Attention and the Luong Attention, were primarily used in tasks like machine translation, where the model learns to align and focus on relevant source words during the translation process [51, 65].

Transformers revolutionized the use of attention mechanisms in neural networks. The transformer architecture, introduced in the paper "Attention Is All You Need" in 2017, presented a novel and powerful approach to sequence modeling [86]. Transformers employ a *self-attention*

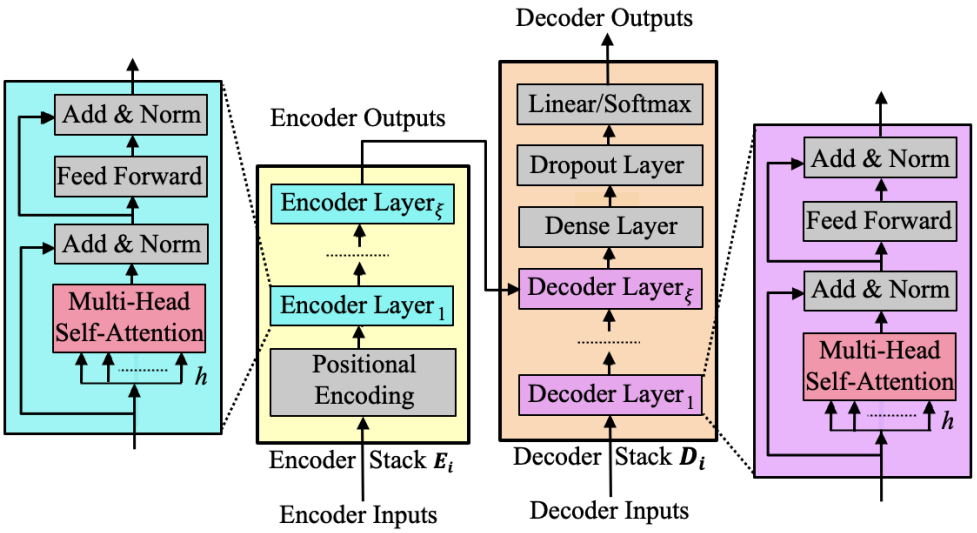



FIGURE 2.3: Structure of a Transformer for the $i$-th user. The architecture contains an Encoder Stack $E_i$ made of $\xi$ Encoder Layers and a Decoder Stack $D_i$ made of $\xi$ Decoder Layers [20] ©2022 IEEE.

mechanism called the "Scaled Dot-Product Attention." This mechanism enables the neural model to effectively capture global dependencies, accurately weigh the significance of elements in the input sequence, and attend to relevant parts in parallel. Consequently, transformers facilitate more efficient and effective learning by surpassing the limitations of traditional RNNs in capturing long-range dependencies. Unlike conventional RNNs and LSTMs, which process data sequentially within fixed memory windows, transformers globally process the input sequence, paving the way for enhanced sequence modeling capabilities. Moreover, transformers employ multi-head attention, executing the self-attention mechanism multiple times in parallel with distinct learned weights. This enables the model to simultaneously consider multiple segments of the input sequence, thus enhancing its capability to capture intricate relationships among the inputs [65]. The parallelization achieved through self-attention results in significantly reduced training times compared to RNNs.

Figure 2.3 illustrates the generic transformer architecture comprising encoder and decoder stacks, which form the neural network developed in this thesis for our trajectory predictors. The *Encoder Stack* consists of a positional encoding layer and several encoder layers, in charge of converting a trajectory to an abstract representation, followed by the *Decoder Stack* to output a predicted trajectory from the learned abstract representations of the user mobility. The *Decoder Stack* consists of several decoding layers, dense layers, dropout layers, and an output layer (linear for regression and softmax for classification problems).

The encoder and decoder layers themselves are composed of multi-head attention modules, add and normalization layers with two residual connections, feedforward layers, and dropout layers. As stated, the self-attention mechanism is a key component of the transformer architecture.

The multi-head attention layer is composed of multiple self-attention units enabling a parallel attention mechanism over distinct parts of the input sequence. In the add and normalization layer, the term *add* refers to the residual connection that prevents gradients from vanishing or exploding during deep neural network training. The feedforward fully-connected layers are responsible for learning the non-linear mapping between the input and output representations, enabling the model to capture complex relationships and patterns. While dropout layers are not typically employed in transformer models due to the presence of regularization techniques like layer normalization and residual connections, they can still be used after multiple dense layers to provide additional regularization and enhance generalization performance, similar to their utilization in other neural network architectures [65].

Transformers utilize projection matrices $W^Q$, $W^K$, and $W^V$ to map the input into the query, key, and value spaces, respectively. In addition, the feedforward network includes parameters $W_1$, $b_1$, $W_2$, and $b_2$. The scaled dot-product attention operation is defined by the equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.14}$$

where $Q$, $K$, and $V$ represent the query, key, and value matrices, respectively, and $d_k$ is the dimensionality of the key vectors. As stated, the attention mechanism is used to weigh the importance of different parts of the input sequence when generating the output sequence.

## 2.5 Unsupervised Machine Learning

Unsupervised learning is an ML paradigm that trains models on unlabeled data, enabling the discovery of hidden patterns and structures within the data. One popular method in unsupervised learning is clustering, which aims to group similar instances together based on their characteristics. In this thesis, our focus is on mobility data, where we employ clustering techniques to group similar trajectories and profile mobile users. Our primary objective is to enhance network scalability. By applying clustering algorithms, we can identify groups of users with similar behaviors, forming homogeneous clusters. This approach allows us to create a single model for each cluster accommodating scalable trajectory prediction techniques.

### 2.5.1 Clustering

To cluster users effectively, it is necessary to measure the similarity between pairs of users' data sequences. This similarity measurement plays a critical role in grouping users based on their characteristics. There are various approaches to quantify user similarity depending on the specific context and data available. In the context of mobility data, measuring similarity often involves utilizing distance metrics such as Euclidean distance or cosine similarity algorithms. In this thesis, we specifically employ the Longest Common Sub-Sequence (LCSS) similarity measure to group users with similar trajectories into distinct clusters [35]. The LCSS measure allows us to capture the common subsequences shared by trajectories. It calculates the similarity between each pair of trajectories in the dataset and generates a symmetric distance matrix. This matrix represents

the variations between trajectories. What sets LCSS apart from other distance measures is its ability to handle sequences of different lengths or alignments. It identifies the longest subsequence that is shared by both trajectories, regardless of disparities in the number of points or temporal alignment.

The mathematical representation of the LCSS similarity measure can be expressed as follows:

$$LCSS(T_1, T_2) = \frac{\text{Minimum Length of } (T_1, T_2)}{\text{Length of the Longest Common Sub-Sequence}} \tag{2.15}$$

where, $T_1$ and $T_2$ represent two trajectories. The Length of the Longest Common Sub-Sequence is the length of the longest subsequence that is common to both trajectories. The Minimum Length of $(T_1, T_2)$ is the minimum length between $T_1$ and $T_2$, considering their different lengths.

After obtaining the distance metrics or similarities for each pair of user data, clustering algorithms can be applied to group the users based on their similarities. In the context of trajectory prediction, where the data consists of sequences of locations, clustering algorithms that rely on the distance between points are particularly useful. In this thesis, we employ various clustering algorithms, such as Birch, DBSCAN, K-Means, Mean-Shift, Ward, and Optics, to group trajectories into distinct clusters based on the distances between their trajectory points [97, 92]. These algorithms compare the pairwise distances between locations within the trajectories and assign them to appropriate clusters. By applying these clustering algorithms, we can effectively partition the trajectories into disjoint clusters, facilitating the identification of distinct groups and patterns within the dataset.

Mathematically, clustering algorithms leverage distance measures to determine the similarity or dissimilarity between data points. Each clustering algorithm has its own mathematical formulation and principles for clustering data based on distance metrics. The K-Means algorithm minimizes the sum of squared distances between data points and their assigned cluster centroids. DBSCAN identifies dense regions by considering the density of neighboring points within a specified distance threshold.

For example, given a dataset with $n$ data points $X = \{x_1, x_2, \ldots, x_n\}$ and a predefined number of clusters $k$, the K-Means algorithm aims to minimize the within-cluster sum of squared distances. This objective is achieved by iteratively updating the cluster assignments and recalculating the cluster centroids. Let $C = \{c_1, c_2, \ldots, c_k\}$ represent the set of cluster centroids. The K-Means algorithm minimizes the following objective function:

$$J(C) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - c_i\|^2 \tag{2.16}$$

where $C_i$ denotes the data points assigned to cluster $c_i$ and $|xc_i|$ represents the Euclidean distance between a data point and the centroid $c_i$. By iteratively updating the cluster assignments and recalculating the centroids, the K-Means algorithm converges to a locally optimal solution that minimizes the objective function $J(C)$.

Other clustering algorithms[1], such as Birch, DBSCAN, Mean-Shift, Ward, and Optics, also utilize mathematical formulations and distance measures to group trajectories into disjoint clusters. Each algorithm has its own specific mathematical representations and principles for clustering based on distance metrics.

## 2.6 Reinforcement Learning

Reinforcement Learning is a subfield of ML concerned with decision-making and learning through interactions with an environment. It involves an agent that learns to make optimal decisions in a dynamic environment by maximizing a cumulative reward signal. Before diving into the details of RL mechanisms, it is important to introduce foundational terms that form the basis of RL: agent, environment, reward, state, action, policy, and Q-Table [65].

- Agent: The agent is an entity that interacts with the environment, learns from its actions, and makes decisions based on its observations and learned policies. It aims to maximize the cumulative reward it receives from the environment.
- Environment: The environment is the external system or framework in which the agent operates. It is a dynamic entity that the agent interacts with and receives feedback from. The environment can be as simple as a simulated game environment or as complex as a real-world system.
- Reward: The reward is a scalar value provided by the environment to the agent after each action. It represents the feedback or evaluation of the agent's behavior. The agent's goal is to maximize the cumulative reward over time by learning to take actions that lead to higher rewards.
- State: The state represents the current situation or condition of the environment. It includes all relevant information that the agent needs to make decisions. The agent's perception or observation of the environment at a particular time is captured by the state.
- Action: An action is a specific decision or choice made by the agent based on its current state. It represents the agent's behavior or response to the environment. The actions available to the agent depend on the specific problem or task.
- Policy: A policy is a mapping from states to actions. It defines the behavior of the agent, specifying the action to be taken in a given state. The policy can be deterministic, where each state is associated with a single action, or stochastic, where the policy selects actions probabilistically.
- Q-Table: The Q-Table is a data structure used in Q-Learning, a popular RL algorithm. It is a lookup table that stores the expected cumulative rewards, Q-values, for each state-action pair ($Q(s, a)$). The Q-Table guides the agent's decision-making process by providing estimates of the long-term rewards associated with different actions in different states. Through

[1] https://scikit-learn.org/stable/modules/clustering.html

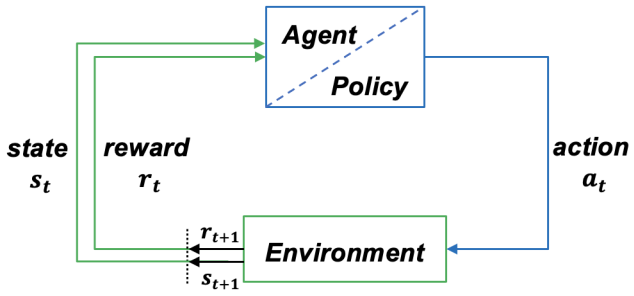



FIGURE 2.4: An overview of Reinforcement Learning.

exploration and exploitation, the agent updates the Q-Table to gradually converge towards optimal policies.

In classic RL, an agent interacts with an environment by taking an action $a_t$ at time step $t$, which affects the environment's state $s_t$, and receives a reward $r_t$ corresponding to the taken action from the environment, as shown in Figure 2.4. The sequence of aforementioned steps is called an episode. The goal of the RL agent is to maximize the expected sum of discounted rewards $\mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$, where $\gamma$ is a discount factor that controls the importance of future rewards. The agent's policy $\pi(a_t|s_t)$ determines the probability of taking each possible action given the current state. The agent learns by iteratively improving its policy through value-based or policy-based methods.

In value-based methods, the agent learns the state-value function $V(s_t)$ or the action-value function $Q(s_t, a_t)$, which estimate the expected sum of discounted rewards starting from the current state or state-action pair, respectively. The agent updates its value function using the Bellman equation:

$$V(s_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})|s_t] \tag{2.17}$$

$$Q(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})|s_t, a_t] \tag{2.18}$$

In policy-based methods, the agent directly optimizes its policy to maximize the expected sum of discounted rewards [65]. The agent updates its policy using the policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t)] \tag{2.19}$$

where $J(\pi_\theta)$ is the objective function that measures the performance of the policy, and $Q^\pi(s_t, a_t)$ is the action-value function under the policy.

Value-based methods are preferable over policy-based methods for NAS in trajectory prediction because they allow for the estimation of the optimal value function, which provides a more

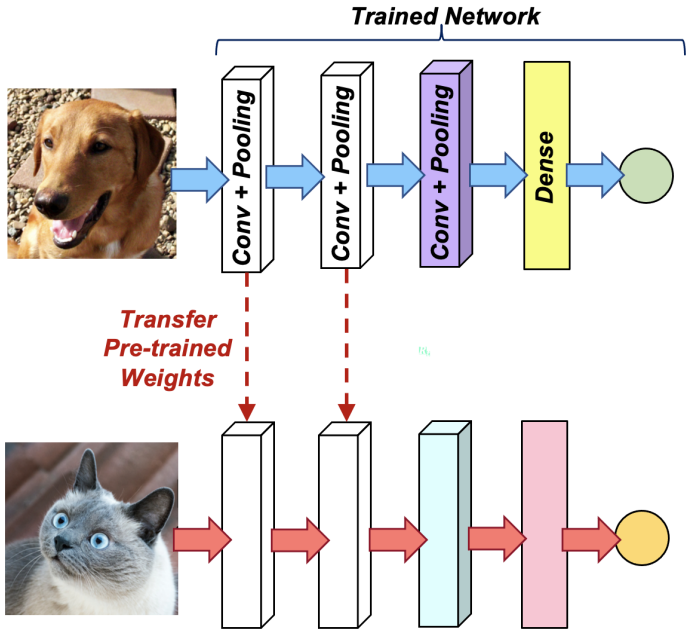



FIGURE 2.5: An overview of Transfer Learning.

accurate measure of the desirability of state-action pairs. This enables the identification of the most promising architectural choices that maximize the expected cumulative reward, resulting in more effective and efficient trajectory prediction models. In this direction, we hire value-based RL in this thesis for the task of reliable and convergent trajectory prediction.

## 2.7 Transfer Learning

Transfer learning is an ML technique that involves utilizing knowledge gained from one pre-trained network to initialize or enhance the learning of another network. The idea behind transfer learning is to leverage the learned representations or weights from a source network, typically trained on a large-scale dataset, and apply them to a target network that is trained on a different but related task or dataset. By utilizing the learned representations from the pre-trained network, transfer learning can expedite the training process and conserve computational resources. This approach is particularly useful when the target task has limited labeled data or when training a model from scratch would be time-consuming or resource-intensive [65].

Mathematically, transfer learning can be represented as follows:

Let $N_{source}$ represent the pre-trained source network with learned weights $W_{source}$, and $N_{target}$ denote the target network with weights $W_{target}$. The transfer learning process involves initializing the weights of the target network $W_{target}$ using the learned weights from the source network $W_{source}$ as follows.

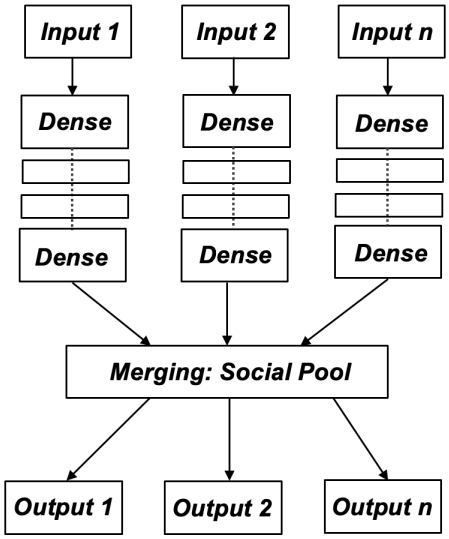



FIGURE 2.6: An overview of Collaborative Learning through Social Pooling.

$$[t]W_{target} = W_{source} \tag{2.20}$$

This initialization step allows the target network to start with a set of well-tuned weights obtained from the source network. Once the target network is initialized with the pre-trained weights, it can be further fine-tuned or trained on a smaller target dataset specific to the task at hand. The transfer of knowledge from the pre-trained network helps the target network to converge faster, generalize better, and achieve improved performance compared to training from scratch.

As shown in Figure 2.5 transfer learning, it is not necessary to transfer the entire network. For example, when training a model to classify different animals, the initial layers that capture general visual features can be shared between tasks such as recognizing cats and dogs. However, in other scenarios where datasets have highly similar characteristics, more layers can be shared to leverage the shared knowledge effectively.

In this thesis, we adopt a transfer learning approach by sharing the first $n-1$ layers of a pre-trained network among users belonging to a similar-trajectory cluster. The reason for sharing $n-1$ layers is that the last layer, responsible for classification, needs to be tailored based on the number of distinct classes in the specific task or dataset. By selectively sharing and reusing layers, we aim to leverage the learned representations, accelerate the training process, and reserve computational resources for trajectory prediction models in a cluster-based setting.

## 2.8 Collaborative Learning

Collaborative learning aims to leverage the collective knowledge and expertise of multiple agents or models to improve individual and overall performance. In this context, each user acts as an independent agent with their own neural architecture, processing their respective inputs. However, at a certain stage, their individual architectures are combined or merged, allowing them to share information and learn from each other. This collaboration enables the models to benefit from the diverse perspectives and insights of each user, leading to improved predictions and performance. This unified representation is then used to generate multiple outputs, each corresponding to a specific user or task [65].

There are several collaborative learning techniques that enable knowledge sharing among multiple agents or models, including Knowledge Distillation, Ensemble Learning, multi-task learning, social pooling, Federated Learning, Multi-agent Reinforcement Learning, game theory, and Cooperative Coevolution. In this thesis, we propose collaborative models that capture the cooperative and non-cooperative interactions among multiple agents and their mutual impact. To achieve this, we employ techniques such as social pooling, federated learning, multi-agent reinforcement learning, and game theory, enabling a comprehensive understanding of the complex dynamics among the agents.

Figure 2.6 illustrates the architecture of a collaborative learning model designed to handle multiple user inputs. The model incorporates distinct neural network architectures for each user, which are subsequently combined using a social pooling layer. The interactions among multiple agents within the model lead to the generation of multiple outputs, capturing the collective intelligence and insights derived from their collaborative interactions. We consider a socio-interactive network comprising $N$ individuals, each contributing input feature denoted as $X_1, X_2, ..., X_N$. These input features capture various aspects of individual behavior, preferences, or characteristics relevant to the prediction task. To harness the collective knowledge and patterns within the social network, we introduce a social pooling layer. The inputs from the individuals, $X_1, X_2, ..., X_N$, are merged using the social pooling operation denoted as Pooling$(\cdot)$. The social pooling layer captures the collaborative information and interactions among individuals, resulting in the generation of aggregated features, denoted as Aggregated Features. The social pooling operation can be defined as:

$$\text{Aggregated Features} = \text{Pooling}(X_1, X_2, \ldots, X_N) = \text{concat}(f(X_1), f(X_2), \ldots, f(X_N)). \quad (2.21)$$

Here, $f(\cdot)$ is a function that maps individual input features to a common feature space, and concat$(\cdot)$ denotes the concatenation operation that combines the transformed features from all individuals. Next, the aggregated features are utilized for predicting multiple output variables. Let's assume we have $M$ output variables to predict. We employ a prediction function, denoted as Predict$(\cdot)$, which takes the aggregated features as input and generates predictions for the multiple output variables. The predicted values are represented as $Y_1, Y_2, ..., Y_M$.

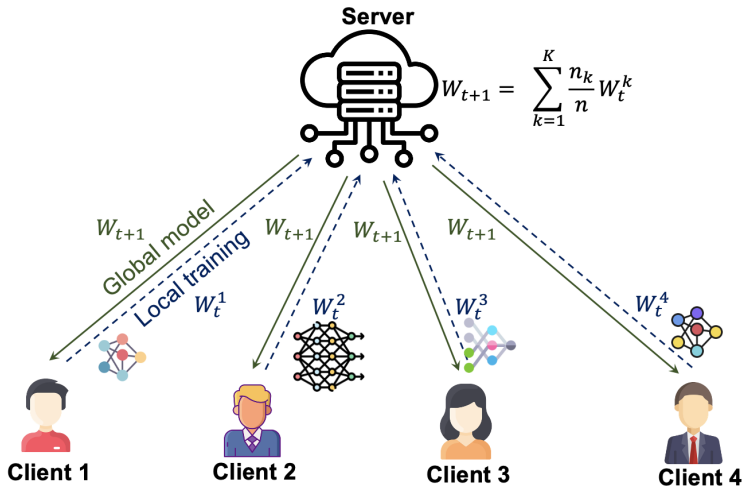



FIGURE 2.7: An overview of Federated Learning.

## 2.9 Federated Learning

As mentioned, federated learning is a form of collaborative learning that involves constructing a single model using data from multiple users. Additionally, federated learning is a distributed model that addresses privacy concerns and enhances scalability compared to centralized collaborative models. By keeping the training process decentralized and performing computations locally on user devices, federated learning ensures data privacy while efficiently leveraging a large number of users' data for model training.

In federated learning, the training process occurs locally on user devices, where each user trains a local model using their own data. The local models are then aggregated to form a global model through a weighted averaging process. The weights of the global model are distributed back to the users, who then repeat the local training with the updated global model. This iterative process continues until the global model converges or achieves the desired performance.

Mathematically federated learning can be modeled as follows: $K$ represents the total number of participating clients, $T$ denotes the total number of communication rounds ($t = 1$ to $T$), $w$ signifies the global model parameters, $w_i$ represents the local model parameters of client $i$, $C_i$ denotes the local dataset of client $i$, $\eta$ represents the learning rate, and $T$ signifies the number of local iterations per communication round. Figure 2.7 shows an example of federated learning among $K = 4$ distributed users. The communication rounds until convergence involve the following steps. Initially, an initial global model $w_0$ is set. Each federated client $i$ performs $T$ local iterations to update its local model parameters $w_{t+1}^i$ using its local dataset $C^i$ and the current global model $w_t$. The local model updates are then communicated and aggregated to obtain the updated global model $w_{t+1}$. This aggregation step involves weighted averaging based on the sizes of the local datasets, ensuring fair representation of client contributions. The weighted averaging can be represented as:

$$w_{t+1} = \sum_{i \in S_t} \frac{C_i}{\sum_{j \in S_t} |C_j|} w_t^i. \tag{2.22}$$

A convergence check is performed, and if a convergence criterion is met, the federated learning process terminates. Finally, the final converged global model $w^*$ is returned. In the local model update step, the optimization algorithm used for updating the local model parameters is denoted by the function Update$(\cdot)$. It can be based on stochastic gradient descent (SGD) or a variant thereof. The convergence criterion in the check step depends on specific requirements and can be based on model performance or other convergence metrics [65].

## 2.10 Game Theory

Game theory is a mathematical framework that analyzes strategic interactions and decision-making among rational agents in competitive or cooperative settings. It provides a powerful tool for understanding how individuals or entities make choices and how these choices impact each other.

In this thesis, the trajectory prediction scenario involves multiple users who are influenced by each other's decisions and may have their own individual desires or objectives. To address this complex and dynamic environment, we further formulate the collaborative learning problem as a non-cooperative game to capture the interactions and strategic behavior of the users. Each user is considered as a player in the game, and they make decisions and adjust their strategies based on the observed outcomes and the strategies employed by other users. The game-theoretic framework enables the users to learn and adapt their strategies over time, taking into account the influence of other users' decisions on their own trajectory prediction.

Before diving into the details of game theory mechanisms, it is crucial to familiarize ourselves with some foundational terms such as a game, players, decision, payoff matrix, Nash equilibrium, dominant strategy, and best response [65].

- Game: A game represents a formal model of strategic interaction between multiple players. It encompasses the rules, actions, and payoffs that define the structure and outcomes of the game.
- Players: Players are the individuals or entities involved in the game. Each player makes decisions or takes actions with the aim of maximizing their own utility or payoff.
- Decision: In a non-cooperative game, each player makes decisions independently, without coordination or communication with other players. These decisions determine the actions taken by the player within the game.
- Payoff Matrix: The payoff matrix is a tabular representation that shows the payoffs or utilities associated with each possible combination of actions chosen by the players. It indicates the rewards or outcomes that each player receives based on their choices and the choices of other players.

- Nash Equilibrium: Nash equilibrium is a concept in game theory that describes a stable state where no player has an incentive to unilaterally change their strategy given the strategies chosen by the other players. In a Nash equilibrium, each player's strategy is optimal, given the strategies of the other players.
- Dominant Strategy: A dominant strategy is a strategy that provides a player with the highest payoff regardless of the strategies chosen by other players. If a player has a dominant strategy, it is always in their best interest to choose that strategy.
- Best Response: A best response is a strategy that yields the highest possible payoff for a player, given the strategies chosen by the other players. It represents the optimal response to the strategies employed by the other players, maximizing the player's own utility or payoff.
- Strategy Profile: A strategy profile in refers to the combination of strategies chosen by all players in a game, representing their individual actions or decisions. It captures the complete set of choices made by the players, influencing the overall outcome of the game. The concept of a best response is closely related to the strategy profile, as it identifies the optimal strategy for a player, maximizing their payoff given the strategies chosen by the other players within the strategy profile.

In a classic non-cooperative simultaneous game, each player chooses their strategy independently of the other players, and there is no communication or coordination between players. Multiple players make decisions simultaneously, without knowing the decisions made by the other players. Let $K = \{p_1, \ldots, p_k\}$ be the set of players, where each player $p_j \in K$ can select a strategy $\sigma_j \in \Sigma$ among $m = |\Sigma|$ possible strategies. The payoff for each player depends on the combination of strategies chosen by all players. The game's outcomes for each player can be represented by a payoff matrix $A \in \mathbb{R}^{m^k \times k}$ with entries $a_{\sigma_1,\ldots,\sigma_k,j}$, where $\sigma_1, \ldots, \sigma_k \in \Sigma$. Each row of matrix $A$ represents one of the $m^k$ possible combinations of strategies for all players. The entry $a_{1,\ldots,k,j}$ represents the payoff for player $j$ when the players choose the strategies $\sigma_1, \ldots, \sigma_k$. The matrix $A$ can be constructed as follows:

$$A = \begin{bmatrix} a_{1,\ldots,1,1} & a_{1,\ldots,1,2} & \cdots & a_{1,\ldots,1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,\ldots,m,1} & a_{m,\ldots,m,2} & \cdots & a_{m,\ldots,m,m} \end{bmatrix} \tag{2.23}$$

After having the payoff matrix, it becomes evident how the decisions of multiple agents influence each other within the game. The payoff matrix reflects the interdependencies and interactions among the agents, showing the rewards or outcomes that each agent receives based on their choices and the choices of other agents. Once we have the payoff matrix, we analyze it to determine the Nash equilibrium.

A strategy profile $\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_k^*)$ is a Nash equilibrium if, for each player $j \in K$, their strategy $\sigma_j^*$ is the best response $b_j$ to the strategies of the other players, i.e., $\forall \sigma \in \Sigma^k : u_j(\sigma^*) \geq u_j(\sigma)$. In this context, $u_j$ denotes the utility function of player $j$, which represents the player's preference over the possible outcomes. In other words, player $j$'s strategy $\sigma_j^*$ is a best response to

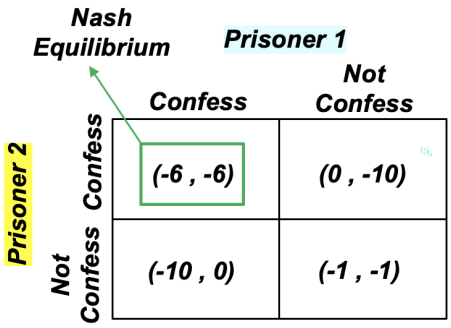



FIGURE 2.8: An overview of Payoff Matrix of a simultaneous Non-Cooperative Game (Prisoners Dilemma).

the strategies $(\sigma_1^*\sigma_2^* \ldots, \sigma_{j-1}^*\sigma_{j+1}^* \ldots, \sigma_k^*)$ chosen by the other players. To find the best response of player $j$, $b_j$, in a non-cooperative game, we use the following formula:

$$b_j(\sigma_{-j}) = \arg\max_{\sigma_j \in \Sigma} u_j(\sigma_j, \sigma_{-j}), \tag{2.24}$$

where $\sigma_{-j}$ denotes the strategies of all other players in the game. To determine whether a given strategy profile is a Nash equilibrium, we can find the best response of each player to the strategies of the other players, using the formula for the best response given above, substituting $\sigma_{-j}$ with $(\sigma_1^*, \sigma_2^*, \ldots, \sigma_{j-1}^*, \sigma_{j+1}^*, \ldots, \sigma_k^*)$. If each player's chosen strategy is the best response to the strategies of the other players in the given strategy profile, then the strategy profile is a Nash equilibrium [65].

Figure 2.8 showcases the payoff matrix of an illustrative example of the simultaneous non-cooperative game known as the *Prisoner's Dilemma*. The Prisoner's Dilemma is a classic game theory scenario where two individuals face a decision to cooperate or betray each other, with the possibility of greater individual gain through betrayal. In this non-cooperative game, the Nash equilibrium may not necessarily yield the global maximum. In the best-case scenario, both players would choose not to confess, resulting in a one-year jail sentence. However, due to the simultaneous nature of the game and the absence of communication between players, the rational decision, equivalent to the Nash equilibrium, is for both players to confess, unless there is a change in incentives and the potential punishment is sufficiently high. We extend the framework of the Prisoner's Dilemma game to model the non-cooperative interactions among competing users in the context of multi-agent trajectory prediction.

## 2.11 Chapter Conclusions

In this chapter, we have explored the theoretical foundations of the mobility predictor, mobility management, predictive machine learning models, and intelligent decision-making techniques

that will be further developed in the upcoming chapters. Through an in-depth examination of the fundamental concepts, methodologies, and techniques underlying these fields, we have established the necessary mathematical and theoretical background to drive our subsequent investigations and analyses.

# Chapter 3

# Related Works

## 3.1 Chapter Introduction

In this chapter, we provide a comprehensive review of the current state-of-the-art in the field of mobility prediction and management, including its applications and use cases and examine the limitations of existing works. In Section 3.2, an overview of existing mobility datasets is provided and compared to the datasets used in this thesis. Section 3.3 compares various handover management and service migration works and algorithms aiming to enhance the latency-sensitive applications in wireless networks. In Section 3.4.1, existing works on isolated-agent trajectory forecasting models are investigated. Social-aware multi-agent trajectory prediction models in dynamic environments are introduced in Section 3.4.2, with their advantages compared to isolated trajectory predictors highlighted. Decentralized multi-agent trajectory prediction models are then presented in Section 3.4.3, discussing their benefits over centralized predictors. Finally, non-cooperative strategic multi-agent trajectory prediction models are delved into and compared to cooperative models in Section 3.4.4.

## 3.2 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 base station's identification related to an event [30, 10]. 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 Call Detail Records (CDR) datasets explore the mobility pattern of individuals logged whenever an user made/received a call or a text message [30, 10]. Gonzalez et al. [30] analyze two datasets with more detailed information about users, such as an average service area of 3 $km^2$ and 30% of base stations covering smaller areas up to 1 $km^2$. Chen et al. [10] provide mobility trajectory reconstruction with GPS coordinates of base stations 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 base station identification and geographical position is not likely to be provided for privacy concerns.

This thesis employs two datasets to support the research, namely the large-scale private dataset from *Orange* telecommunication S.A.[104], and the small-scale public dataset from *ETH*[68] plus *UCY* [47]. More information regarding these datasets is provided in Section 4.4. To protect user privacy, the Orange dataset anonymizes user and cellular base station identifiers, as well as removing GPS data. Consequently, user trajectory forecasting becomes more challenging due to the absence of geographical coordinates. In contrast, the ETH+UCY datasets offer GPS coordinates of pedestrians obtained from bird-eye cameras.

## 3.3 Mobility Management in Wireless Networks

Effective handover management and service migration in wireless networks rely heavily on efficient mobility management, particularly in new-generation mobile networks, e.g., 5G and 6G, where dense cellular networks and high mobility are common. The QoS experienced by end-users is directly affected by the performance of mobility management, especially in modern applications where even minor delays can have significant consequences. To address this, several approaches to mobility management have been proposed in the literature, including both ML-based and non-ML-based schemes.

### 3.3.1 Handover Optimization

Authors in [93] propose a delay-oriented cross-tier handover skipping scheme, i.e., a handover algorithm that minimizes the number of handovers 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 handover execution and proposed a resource allocation scheme in target base stations to reduce blocking probability. However, it does not employ predictive schemes or mobility information into the decision, which may improve the decision quality and positively impact user QoE.

Gong *et al.* [29] propose a multi-criteria handover 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 *analytic hierarchical process* attributes weights to each parameter based on predefined degrees of importance. Based on these schemes, the cross-tier handover performance is improved in terms of failure probability and ping-pong rate, i.e., when a user disconnects and connects multiple times from a base station or a group of base stations in a short period of time. To make the decision more reliable and avoid unnecessary handovers, the authors also consider previous measurements in the process. However, their proposed solution is not as fast-reactive as necessary on challenging scenarios of modern networks.

Authors of [55] and [59] provide prediction-based handover optimization mechanisms with limitations. The authors in [55] 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 handovers during a user's path in a small cell network. However,

when considering a small cell network, a higher number of handovers is expected to keep service and coverage continuity for the users, thus simplifying mobility prediction and a mechanism to avoid handover execution may not be suitable for all scenarios. The authors of [52] propose a group-based pre-handover authentication scheme for 5G high-speed rail networks, which has shown some success. However, the performance of the mobility predictor used in the scheme has multiple limitations. In all above cases, a more accurate, efficient, and reliable prediction model is essential for a good handover algorithm.

### 3.3.2 Service Migration

The process of decision-making for service migration in MEC environments has presented significant challenges that require large-scale experiments and greater utilization of AI techniques, as noted in [102]. Many works focus on service pre-migration in vehicular scenarios, such as Yu *et al*. [95], 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.

Jing *et al.* [39] propose LSTM-based prediction of cloud resources for service migration purposes without considering knowledge of mobility prediction. Zhang *et al.* [100] proposes a deep-belief network for prediction of tasks in a cloud computing environment. However, prediction techniques are far too simplistic for modern networking scenarios. Thus, modern 5G 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.* [49] showcases the opportunities for optimizing resource allocation in a wireless-access edge computing network. The authors propose 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 a significant decrease in QoS.

Furthermore, Zhang *et al.* [101] 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.* [48] propose a joint service migration and caching for a Service Defined Network (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 that dynamically learns a caching strategy. However, the system does not make use 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. In the same manner, [102] present a survey on decision-making in the context of task migration. Such a mechanism, however, is designed for small task offloading from mobile devices and can be adapted for larger containers and VMs on demand in the context of edge computing.

On the other hand, Ouyang *et al.* [66] 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. However, it is not fully successful with respect to works that consider the user mobility. Gao *et al.* [26] as well 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.

Currently, there is a research gap in the development of accurate, robust, and personalized trajectory predictors for effective mobility management in wireless networks. To address this gap, we propose a high-performance RL-designed LSTM NN-based mobility predictor. Utilizing this RL-LSTM trajectory prediction, we then develop the Reinforcement Learning-based Handover for Edge Computing (RL-HEC) system, which combines RL techniques into the handover decision process. Additionally, we introduce the Reinforcement Learning-based Service Migration (RL-SM) system, which leverages individual user mobility prediction to perform proactive service migrations, ensuring continuous service. Integrating TP and mobility management in this way ensures guaranteed uninterrupted communication, QoS, and service migration performance, resulting in improved overall performance and reliability of wireless networks. By combining advanced DL techniques with traditional mobility management approaches, we aim to provide a more comprehensive and effective solution for managing mobility in wireless networks.

## 3.4 Mobility Prediction

### 3.4.1 Isolated-Agent Trajectory Prediction Models

TABLE 3.1: Comparison of the state-of-the-art isolated-agent trajectory predictors [21] ©2022 IEEE

| Trajectory Predictor | Model | ANN | RL | Clustering | TL |
|---|---|---|---|---|---|
| Zhang et al. [103] | RNN | ✓ | - | - | - |
| Phillips et al. [70] | LSTM | ✓ | - | - | - |
| Nikhil et al. [64] | CNN | ✓ | - | - | - |
| Shrivastava et al. [79] | LSTM | ✓ | - | ✓ | - |
| Sung et al. [81] | Markov | - | - | ✓ | - |
| Karimzadeh et al. [41] | LSTM | ✓ | ✓ | - | - |
| RC-TL | CNN | ✓ | ✓ | ✓ | ✓ |

Trajectory forecasting is a critical component in intelligent mobility and transportation services. In general, TP can be categorized into two classes: physics-based and maneuver-based models [45]. Physics-based prediction models are mainly based on statistical models such as the

family of Bayesian filters and Kalman Filters (KF). The authors in [8] implement KF to predict future trajectories of mobile users. On the other hand, maneuver-based predictions are mainly based on ML algorithms [91]. Heuristic-based classifiers [37], Markov models and Random Forest (RF) classifiers are all examples of maneuver-based trajectory predictions through different ML algorithms.

Traditional mobility prediction methods such as Markov-based, Hidden Markov models, Gaussian models, and Kalman filters have limitations in accurately predicting user movement patterns in modern networks that generate large volumes of data [72, 89, 71]. Markov models, for instance, assume that the future state of a system depends only on the current state and not on its history. Hidden Markov models assume that the underlying state of the system is hidden and can only be inferred from observable data. Similarly, Gaussian models assume that the distribution of data follows a normal distribution, which is often not the case for mobile user location data. Kalman filters rely on the assumption that the system is linear and has Gaussian noise, which limits their applicability in many real-world scenarios.

In this direction, ML and DL techniques have been proposed to improve TP accuracy. In recent years, AI algorithms have replaced conventional statistical models to analyze mobility data and predict future trajectories of mobile users, e.g., pedestrians and vehicles [16, 76, 84]. In the literature, the problem of TP has been extensively tackled with RNN [103], and their variants including GRU [13] and LSTM [36, 70, 69], as successful DL models designed for time-series prediction. Despite the success of RNNs, GRUs, and LSTMs they face a slow training process, as they must process the input data in sequential order. In contrast, CNNs are other powerful DL tools that can concentrate on sequential data from a hierarchical perspective processing the data as a whole and, thus, can become a suitable alternative for RNNs. The predictors presented by Nikhil et al. [64] and Zamboni et al. [98] is from the limited existing works that apply CNNs for the isolated TP field.

Although all the abovementioned DL works successfully predict trajectories, they design the NN architectures manually following human-expert-based heuristics, which is an error-prone and time-consuming process. NN architecture design is an NP-hard problem that conventional optimization methods, such as random search, cannot solve it in polynomial time as the search space expands. Moreover, designing NN models can be especially challenging in less-explored domains such as TP, compared to more established fields such as image processing, where a wealth of knowledge is readily available for NN design. Furthermore, they apply the same NN architecture to every user dataset, assuming everyone has similar mobility behaviors without personalization. Several search methods have been proposed to automate the NN architecture design process given a dataset in other fields than mobility prediction. In their work, Elsken et al.[17] conduct a survey on existing NAS methods and demonstrate that formulating hyper-parameter optimization as a Reinforcement Learning (RL) algorithm is a more efficient search technique when compared to naive methods like Grid Search (GS), Random Search (RS) [9], and Bayesian search [77, 43]. Bayesian optimization leverages probabilistic models to search for optimal hyperparameters, reducing the need for exhaustive evaluations. However, constructing a probabilistic model of the hyperparameters and iteratively updating it with new observations is computationally expensive, specifically when dealing with large search spaces. Zoph et al. [105] were the first to introduce the concept of computationally-efficient NN architecture design through RL. They used an RNN

controller trained with the REINFORCE policy gradient algorithm to generate various NN architectures on the CIFAR-10 image dataset. Most of the NAS works apply RL to find the best architecture for image classification tasks, where defining well-suited search spaces is comparatively easy due to human experience and the existence of several manually-designed models [105, 6, 40].

Yet, the potential of NAS methods in less explored domains is still unclear [17]. Inspired by Zoph et al. [105], the use of RL in trajectory prediction has been suggested by Karimzadeh et. al. [41], but the current approach raises concerns regarding its efficiency. In the existing methodology, the complete set of epochs (i.e., 200) for each neural architecture is trained in every RL episode. This mixing of grid searches, which can identify global optima, and RL's ability to narrow down the search space early, is inefficient. Consequently, this model does not yield polynomial time complexity and leads to a significant waste of computational resources. Therefore we propose and study a convergent RL-based NAS method in the field of trajectory prediction by proposing to train the suggested LSTM, on each RL episode, with only partial training epochs (e.g., 10 out of 200) [104].

Moreover, Computationally-light NAS methods have not been studied in the literature [17]. RL for NAS is computationally lighter than GS methods and can achieve better accuracy than RS methods, but is still computationally expensive. In a real network scenario with thousands of mobile users' trajectories, training an RL agent per user dataset to optimize their neural network architecture would be impossible. Multiple users in an urban area might partially follow similar trajectories during a specific period of the day by chance or by groups' intentions [11]. In this direction, we propose to cluster similar trajectory users and train a single RL agent for each cluster of users instead of for each user to reduce computational requirements in large-scale individual TP networks.

In the literature, some relevant TP works have suggested clustering similar users. Nevertheless, their approaches have many shortcomings and are not within the scope of NAS. For example, the recently proposed model in [79] clusters similar trajectory users and then feeds only parts of complete trajectories (referred to as partial trajectories) within a cluster to an LSTM. However, their approach faces three problems. First, training only a part of trajectories disregards long-term spatio-temporal dependencies information. Besides, training partial trajectories of all users within all clusters still requires a considerable amount of computational resources. Moreover, Shrivastava et al. [79] apply the same heuristically-designed LSTM architecture to all clusters, which does not guarantee the optimal prediction performance.

Alternatively, Sung et al. [81] cluster similar users and then aggregate all user data within a cluster by averaging them to a single data sequence and feeding it to a Markov-based predictor. This model also suffers from three shortcomings. Primarily, averaging all users' data discards useful spatio-temporal information. Furthermore, their model needs access to all users' data for the aggregation, which is expensive in terms of communication and computation. Finally, Markov-based predictors are much less potent than ANNs and cannot guarantee optimal performance.

In this direction, we propose Reinforcement Convolutional Transfer Learning for Large-scale Trajectory Prediction (RC-TL) system [21] to address all the shortcomings mentioned above by bringing a combination of NAS and TL to the cluster-level mobility prediction. RC-TL offers an

accurate yet computationally efficient trajectory prediction without the requirement of processing all users' data.

Table 3.1 compares our solution RC-TL to existing state-of-the-art isolated pedestrian or vehicular trajectory predictors in terms of the techniques used to achieve the mobility prediction. We can observe that the majority of works use some form of ANN as a predictor. We observe RC-TL is the only compared work that employs an RL-designed ANN together with TL and clustering techniques to achieve higher accuracy at comparable or lower computation costs.

Compared to our previous work on RL-LSTM trajectory prediction in [104], where the neural network was personalized for each individual user, RC-TL takes a more realistic approach and is scalable by personalizing a single neural architecture for a cluster of neighboring trajectories. However, RC-TL does not consider social interactions between users and only predicts trajectories based on isolated data from each individual user.

### 3.4.2 Social-aware Multi-Agent Trajectory Prediction Models

TABLE 3.2: Comparison of the state-of-the-art multi-agent social-aware trajectory predictors [20] ©2022 IEEE

| Trajectory Predictor | Neural Network | Social Module | NAS |
|---|---|---|---|
| Alahi et al. [3] | LSTM | Social Pool | - |
| Gupta et al. [31] | LSTM-GAN | Social Pool and Pooling Vector | - |
| Sadeghian et al. [74] | LSTM-GAN | Attention | - |
| Yu et al. [94] | Transformer | Graph Convolution | - |
| INTRAFORCE | Transformer | Social Pool and Clustering | RL |

Social-aware predictors leverage the social interactions among individuals to improve TP performance by considering group intelligence. Data-driven social mobility predictors have gained popularity compared to the previously proposed *Social-force* models, which use simple repulsive and attraction forces [33]. The vast majority of modern social-aware trajectory predictors are based on DL models, such as RNNs, LSTMs, CNNs, and attention-based neural networks, which require less computation and achieve higher prediction accuracy compared to social-force models due to their better modeling of sequential patterns [104, 21, 28]. Instead of modeling kinetic forces and energy potentials as in social-force models, social-pooling [3, 31], attention [4, 74], and graph [94, 60] mechanisms complement neural networks to share information about neighboring user's trajectories to capture complex interactions in crowded environments.

Alahi et. al. [3] introduce the Social-LSTM, which relies on a social pooling layer that connects multiple LSTMs to model the interaction among pedestrians. Social-LSTM considers user interactions within a fixed-size local area, which reduces the flexibility in considering the interaction among any two users that might affect each other's trajectory. To solve such limitation, Social-GAN [31], a Generative Adversarial Network (GAN)-based trajectory predictor, extracts social interactions among all users in the system. Social-GAN stores the relative positions between all users

of the social pool in a *pool vector*, in order to let each user make mobility decisions based on information of other users at the decoder side. However, Social-GAN weights each user trajectory's influence on the model identically. Sophie [74] solves this limitation by proposing another GAN-based social network that feeds all users of the scene to a social attention component, which aggregates various agents interactions and extracts users who have more influence on each other from the surrounding neighbors. Even though the social-pooling algorithms perform well in learning and predicting social interactions, they are computationally complex and resource-intensive because they feed trajectories of all users within a scene to the social predictor's encoders. In contrast to these works, our propose Intra-Cluster Reinforced Social Transformer for Trajectory Prediction (INTRAFORCE) system, which generates a set of Social-Transformer trajectory predictors, each trained on the data of a different group of users with similar mobility. Transformers (TF) have emerged as a state-of-the-art NN architecture for time series prediction. The TF architecture was first introduced in the seminal paper "Attention Is All You Need" [86], which showed that transformers can achieve state-of-the-art performance on time-series machine translation tasks. In this direction, INTRAFORCE adapts TFs for social trajectory prediction models. For an environment containing $n$ mobile users, INTRAFORCE learns the social interactions among the $n_k \ll n$ users in each cluster $c_k$. In this way, the cluster's social-transformer is trained over $n_k$ datasets instead of $n$, saving the computational resources needed for training over the $n - n_k$ datasets associated to users whose mobility features are irrelevant for cluster $c_k$.

Another shortcoming of the existing social trajectory predictors, similar to isolated models in Section 3.4.1, is that their NN architectures are heuristically designed, and that the same NN is applied to a wide range of mobility datasets with no neural architecture adaption. To mitigate such drawbacks, we propose to personalize the NN architecture design given each group of neighbor users' motion behaviors. We propose to formulate the problem of optimizing NN hyper-parameters of the multi-agent social-aware mobility prediction through RL. Users within each cluster share similar trajectories and exhibit stronger interactions with each other. Therefore, a RL algorithm is proposed to personalize a NN for each cluster. Previously, in RC-TL [21] we suggested applying RL for individual TP, where users are left isolated. While, in INTRAFORCE we propose and develop a RL algorithm in the field of social TP. Table 3.2 compares the characteristics of our solution with those of existing state-of-the-art social-aware trajectory predictors.

### 3.4.3 Decentralized Multi-Agent Trajectory Prediction Models

As presented in Section 3.4.2, Social-LSTM [3], Social-GAN [31], Sophie [74], Social-Ways [4], Social-STGCNN [60], and STAR [94] are popular social-aware TP models in the existing literature. However, all of these works are based on centralized models, which can give rise to scalability and confidentiality issues. FL models are proposed to enable decentralized ML, allowing multiple devices to collaboratively train a shared model without exchanging raw data to preserve data privacy and reduce the network overhead by minimizing the amount of data sent to a central server.

FedAvg [57] is the first federated training method designed for training distributed NNs that uses Stochastic Gradient Descent (SGD) optimizer to train each federated client. To improve the

TABLE 3.3: Comparison of the state-of-the-art multi-agent federated trajectory predictors [18] ©2023 IEEE

| Characteristics | Flow-FL [54] | ATPFL [87] | MOB-FL [90] | FedForce |
|---|---|---|---|---|
| Improvement over FedAvg | ✓ | ✓ | ✓ | ✓ |
| Training participant election | ✓ | - | - | ✓ |
| Local data quality estimation | - | - | - | ✓ |
| NN architecture optimization | - | ✓ | - | ✓ |
| Computational optimization | - | - | ✓ | ✓ |
| Communication optimization | - | - | - | ✓ |
| Multi-objective RL | - | - | - | ✓ |

classical FedAvg, Per-FedAvg [22] allows federated users to enforce an extra SGD update after receiving the global model at the end of training rounds to add a small amount of personalization to each client's model. To further advance the FL performance, pFedMe [82] suggests using diverse optimization methods letting federated users perform multiple extra SGD updates as far as each personalized model does not diverge noticeably from the global model parameters. Inspired by above models, researchers have studied FL-based TP. PMF [23] proposes a group optimization approach to advance the conventional FL algorithm for human mobility prediction achieving a better tradeoff between prediction performance and privacy. Flow-FL [54] proposes a gossip learning, server-less FL, model for robot TP that studies the effects of data collection-related parameters on the FL's performance. Despite the success of forenamed works in FL-based TP, they suffer from resource-intensive computations on the client side since there is no limit on the upper bound number of FL participants. Moreover, none of these TP works estimates local users' data quality to be able to choose eligible users for training. In contrast, we propose Network-adaptive Federated Learning for Reinforced Mobility Prediction (FedForce) system, which trains a portion of clients that their local data are qualified before participation based on their computed regularity ratios. Moreover, training a set of users during FL rounds instead of all users lets FedForce save considerable computational resources. Some of the FL studies explore complex client selection strategies [12]. However, our proposed regularity ratio offers a light weight solution. Our method is highly efficient and fast to compute on each local device. Moreover, our approach transmits only a single value over the wireless link representing the user's trajectory information to the server, resulting in excellent communication efficiency and minimal overhead.

To initialize the federated local training process, the abovementioned TP papers adopt a heuristically designed NN architecture based on the experts' domain knowledge, which is time-consuming and error-prone, similar to the problem of centralized TP works stated in Section 3.4.1 and Section 3.4.2. Manually-designed models are flawed especially in less-explored domains such as TP compared to the image processing field, where there is small information available about manually-created models compared to the image processing field. A recent federated TP study, ATPFL [87], proposes to use the Auto-ML-based NAS to automate the neural architecture design process. However, ATPFL considers only the predictor's performance to be optimized through NAS, ignoring clients' computational and communication resource constraints. On the other

hand, MOB-FL [90] is the only FL-based mobility predictor that considers the computational resource utilization of constraint client devices. MOB-FL proposes an accelerated FL algorithm to maximize the resource utilization of intelligent connected vehicles over short-lived wireless connections by optimizing the duration of federated training rounds and the number of local iterations. However, this acceleration approach is simply fine-tuning the ML training process that considers neither the performance of the predictor nor the size of the model which must be transmitted over the wireless network. Therefore, there have not been many federated TP researches taking care of predictors' performance, local clients' resource limitations, and the communication throughput. In contrast to previous works, we propose FedForce, a multi-objective RL-based NAS for federated TP to address this research gap. Table 3.3 shows that FedForce addresses a wider range of issues from prediction accuracy to network performance compared to state-of-the-art FL-based TP works.

Unlike our previously proposed RC-TL [21] and INTRAFORCE [20] trajectory predictors, in Section 3.4.1 and Section 3.4.2, that only rewarded high-accuracy neural networks, FedForce rewards architectures that optimize multiple objectives simultaneously, including accuracy, model size, training time, and transmission time.

### 3.4.4 Strategic Multi-Agent Trajectory Prediction Models

TABLE 3.4: Comparison of the state-of-the-art multi-agent cooperative and non-cooperative trajectory predictors [19] ©2023 IEEE

| Characteristics | Alahi [3] | Bahram [5] | Geiger [27] | Ma [53] | GTP-Force |
|---|---|---|---|---|---|
| Social-aware TP | ✓ | ✓ | ✓ | ✓ | ✓ |
| Personalization | - | - | - | ✓ | ✓ |
| RL-designed NN | - | - | - | - | ✓ |
| Decentralized TP | - | - | - | - | ✓ |
| Strategic TP | - | ✓ | ✓ | ✓ | ✓ |
| Non-Cooperative Game | - | - | ✓ | ✓ | ✓ |
| Simultaneous Game | - | - | ✓ | - | ✓ |
| Sequential Game | - | ✓ | - | ✓ | - |

Social-aware TP models face a significant challenge beyond NN architecture inflexibility and centralized models. These works assume cooperative behavior from all users in a multi-agent environment. However, in reality, human agents tend to optimize their personal goals instead of joint strategies, leading to significant inaccuracies in joint TP. According to the survey conducted by Rudenko et al. [73], classical AI and game-theoretic approaches hold great potential for modeling human behaviors in multi-agent systems. As a step in this direction, Ma et al. [53] and Geiger et al. [27] are among the few existing papers that partially address the problem of social-aware non-cooperative TP through Game Theory (GT). Bahram et al. [5] propose a cooperative prediction and planning framework for automated driving in dynamic environments based on

the game theory methods. However, this work does not consider the non-cooperative dynamics among mobile users. Ma et al. [53] propose a method for predicting the interactive dynamics of pedestrians using a combination of GT and deep learning-based visual analysis via Fictitious Play. In Fictitious Play, players play their best responses to their opponents. Each player updates their beliefs about the other players' strategies based on the observed outcomes and then selects their strategy for the next round. However, a limitation of this approach is that Fictitious Play is designed for sequential games and may not be well-suited for modeling simultaneous games. In the context of TP problems, individuals may take actions simultaneously, necessitating the use of non-cooperative simultaneous games to more accurately model their impulsive behaviors. On the other hand, Geiger et al. [27] propose a game-theoretic framework for predicting the future trajectories of multiple agents in a social environment, using implicit layers to learn the best response of each agent in a Nash equilibrium of the game. The implicit layer is a neural network layer that learns the underlying relationships between input and output data through a non-linear mapping function. However, this approach lacks personalized neural network models for individual agents to capture their unique characteristics and identify the best trajectory predictions. Personalized models can help to capture the unique features and preferences of each agent, which can ultimately lead to more accurate predictions. In contrast, the use of a single implicit layer to calculate the expected utility of different TPs may not be sufficient to capture the full complexity of the social environment and the individuals within it.

In contrast, we propose Game-Theoretic Trajectory Prediction through Distributed Reinforcement Learning (GTP-Force) system, which addresses the limitations of the abovementioned methods by taking a number of steps. First, GTP-Force decentralizes the training process on local devices, while the aggregation step is performed through a centralized server. Second, GTP-Force personalizes the NN architecture for a cluster of similar-trajectory users based on their intra-cluster unique mobility data features, using a RL algorithm. Third, GTP-Force defines a non-cooperative simultaneous game to search for the best combination of NN models among competing inter-cluster users. Table 3.4 shows that GTP-Force addresses a wider range of issues from prediction accuracy to network performance compared to state-of-the-art game theoretic TP works.

Unlike our previously proposed INTRAFORCE [20] and FedForce [18] trajectory predictors, GTP-Force focuses on modeling non-cooperative behaviors among inter-cluster users. This approach is justified because inter-cluster users often have different data features and computing requirements compared to intra-cluster users.

## 3.5 Chapter Conclusions

In this chapter, we discussed the previous research conducted in the same field as our thesis. This provided insights into the significance of our research and identified some limitations in existing works. Our proposed models, which we will describe in detail in the following sections, aim to address these challenges.

# Chapter 4

# Overview of Mobility Scenarios, Datasets, and Evaluation Metrics

## 4.1 Chapter Introduction

This chapter serves as a foundation for the subsequent chapters where we develop various trajectory predictors. It focuses on the exploration and definition of mobility scenarios we have considered, specifically tailored for mobility prediction and management, enabling the development of the proposed technologies.

Within this chapter, we introduce the datasets used in this thesis. Additionally, we discuss the techniques employed for data preparation and preprocessing, optimizing the input for our AI predictors. Furthermore, we present the experimental setups and evaluation metrics that will be utilized to assess the performance of our proposed trajectory predictors in the next chapters.

Apart from introducing the analytics, datasets, preprocessing, and experimental setups, the main contribution of this thesis is the presentation of our proposed regularity ratio metric. This metric, based on signal processing in the time and frequency domains, allows us to estimate the quality and periodicity of user data without relying on an AI predictor.

## 4.2 Mobility Prediction Scenario

We define a scenario in which $n$ users move within an urban area containing $S$ base stations providing Internet access via a cellular radio network. Each user has a wireless device that connects to the base station with the strongest signal. As users move, the received signal power from the base stations changes, which requires a *handover* to a new base station. The timestamps at which each user connects and disconnects from each base station are recorded. In our system, without loss of generality, we assume that time is discretized as a sequence of countable time slots of variable or uniform length, that we index with the symbol $t \in \mathbb{N}$. We assume that at any timestamp $v_u$, user $u$ is located at *coordinates* $(x_u, y_u) \in \mathbb{R}^2$ and may be connected to *a base station* with *cell ID* $b_u \in \mathbb{N}$. The vector $p_u = (v_u, x_u, y_u, b_u) \in \mathbb{R}^3 \times \mathbb{N}$ represents a single data point about the user's status and is referred to as the user *information vector*. This formulation allows for cases in which information

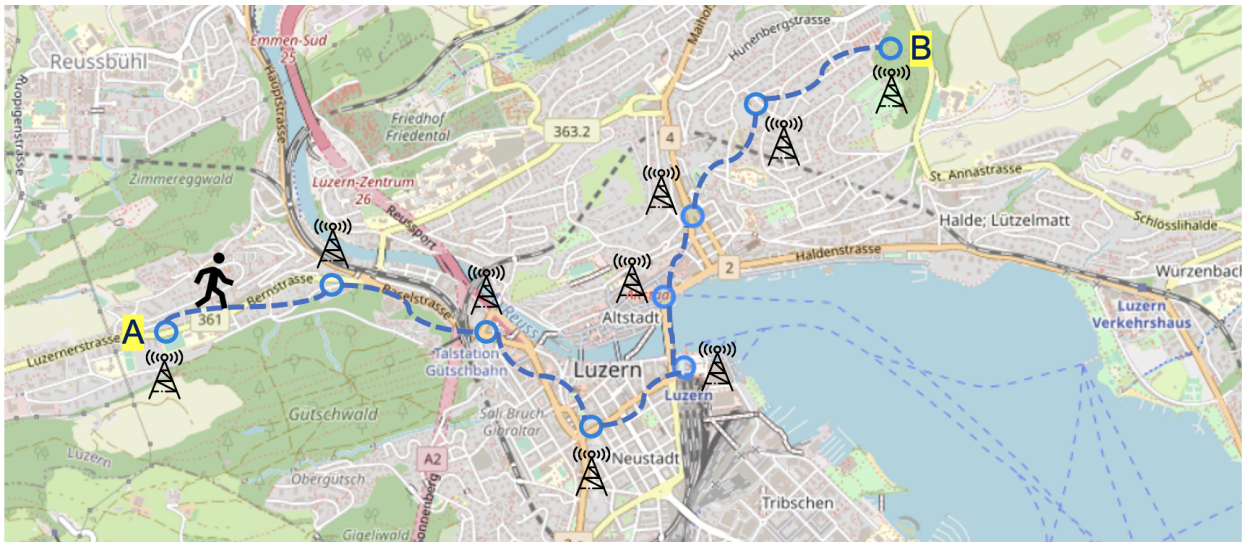



FIGURE 4.1: Trajectory prediction as a classification task. The map images presented in this thesis were created using Folium Python library for visualization purposes and does not involve public map data sources.

about the base stations or the exact locations of users may not be available. Each user $u$ has a total of $m_u$ user information vectors recorded. The set of these vectors $T_u = \{p_u(1), \ldots, p_u(m_u)\}$, is referred to as the user's trajectory, and the set of all user trajectories is denoted as $\Theta = \{T_1, \ldots, T_n\}$.

The set of user trajectories can be partitioned into a set of disjoint clusters $C = \{c_i \subseteq \Theta | i = 1, \ldots, q\}$, with $c_i \cap c_j = \emptyset, \forall i, j \in \{1, \ldots, q\}, i \neq j$, where each cluster contains close-distance trajectories. Our proposed models operate on each cluster independently, thus, an instance of the proposed method runs in parallel for each trajectory cluster.

Given the mobility scenario, RC-TL aims to predict the future location $p_u^{t+1}$, and subsequently, future trajectory $T_u^{t+1}$, of an individual user $u$, using its $m$ past location data $\{p_u^t, p_u^{t-1}, \ldots, p_u^{t-m}\}$. On the other hand, INTRAFORCE, FedForce, and GTP-Force predict the future trajectory $T_u^{t+1}$ of each user, $u$, based on the mobility data of both the individual and other users $\{\Theta^t, \Theta^{t-1}, \ldots, \Theta^{t-m}\}$, utilizing multi-agent models' collective intelligence. Intra-cluster features are utilized to personalize the models in INTRAFORCE and FedForce for centralized and decentralized training, respectively. GTP-Force, on the other hand, personalizes multiple non-cooperative NN architectures based on inter-cluster user adversaries.

The task of ML-based trajectory predictors involves training a model to learn the patterns of movement and predict future trajectories of mobile users. This process typically involves dividing a dataset into *training*, *testing*, and *validation* sets. The training set is used to train the model by feeding it a portion of the data and allowing it to learn the underlying patterns of movement. The testing set is then used to evaluate the performance of the model by testing its accuracy in predicting future trajectories. Finally, the validation set is used to further evaluate the performance of the model and make any necessary adjustments to improve its accuracy. The validation set plays a crucial role in ensuring the generalization of the model by assessing its performance on


FIGURE 4.2: Trajectory prediction as a regression task. The map images presented in this thesis were created using Folium Python library for visualization purposes and does not involve public map data sources.

unseen data, aiding in the identification of potential overfitting or underfitting issues. By splitting the dataset into these three sets, the ML model can learn and predict future trajectories accurately, leading to more effective mobility management in wireless networks.

Trajectory prediction can be approached as both a *classification* and a *regression* task, depending on the specific problem and the desired output format.

### 4.2.1 Mobility Prediction as a Classification Task

In the case of cellular networks, mobility is often represented by a sequence of connected base station IDs. As users move from one location to another, they connect to different base stations, and this sequence of base station IDs represents their mobility. In this case, the user information vector can be defined as $p_u = (t_u, b_u)$ . By analyzing this sequence of IDs over time, the trajectory of the user can be derived, and their movement patterns and behaviors can be studied. In this case, the task of trajectory prediction becomes a classification problem. The goal is to predict the next base station ID that the user will move to. This approach is commonly used in cellular networks to optimize handover procedures and reduce call drops. ML algorithms can be trained on the historical base station IDs visited by the user to predict their future movements accurately. Figure 4.1 shows an example of a trajectory defined as a sequence of cellular base stations.

### 4.2.2 Mobility Prediction as a Regression Task

In location-based services, the historical sequence of visited locations is represented by GPS coordinates. Each time a user visits a new location, the GPS coordinates of that location are recorded, creating a sequence of visited locations. In this case, the user information vector can be defined

as $p_u = (t_u, x_u, y_u)$. By analyzing this sequence of GPS coordinates over time, the trajectory of the user can be derived, and their movement patterns and behaviors can be studied. In this case, the task of trajectory prediction becomes a regression problem. The goal is to predict the next GPS coordinate, which includes both longitude and latitude. This approach is commonly used in location-based services to predict user movements and to recommend relevant services based on their location. ML algorithms can be trained on the historical GPS coordinates visited by the user to accurately predict their future movements. Figure 4.2 shows an example of a trajectory defined as a sequence of GPS coordinates.

## 4.3 Mobility Management Scenarios

### 4.3.1 Handover Management

We consider a mobile network, such as the one represented by Section 4.2. We assume the presence of $N$ User Equipments (UEs), each connected to one of the $B$ base stations present in the scenario. Users move through the scenario and trigger handover events according to the handover algorithm. A given UE $n \in N$ may or may not have a mobility prediction model $m$ associated with it, depending on the quality $Q$ of the data available for the UE. Each model $m \in M$ has an accuracy level known by the network. Overseeing the handover process, we consider a handover Manager similar to the 4G Mobility Management Entity (MME) or the 5G Access and Mobility Management Function (AMF). This handover Manager has access to the individual mobility prediction models for users and their connection history for ping-pong detection.

### 4.3.2 Service Migration

Let $S_{n,b}$ denote the service being consumed by user $n$ at base station $b$, and let $E_b$ denote the edge data center associated with base station $b$. We assume that each edge data center is capable of running cloud-based services for users in the scenario. To optimize the services being consumed at the network's edge, we can use learning models at the edge of the network. Let $M_{b,k}$ denote the learning model associated with edge data center $E_b$ for optimizing services at base station $k$. 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. To perform service migrations proactively based on user mobility prediction, we can use a next base station (BS) prediction scheme. Let $B_n^{t+1}$ denote the next base station that user $n$ will connect to based on the mobility prediction scheme.

## 4.4 Datasets

Location-based mobility datasets are important resources for studying human and vehicle mobility patterns in urban environments. These datasets contain spatiotemporal information about the locations visited by individuals or vehicles, which can provide insights into the dynamics of urban mobility. The location information can be collected through various means, including GPS latitude-longitude coordinates, cellular base station signals, or Wi-Fi access points. These datasets


FIGURE 4.3: An example of mobile users' visited locations (connected base stations) based on the private Luzern telecommunication dataset. The map images presented in this thesis were created using Folium Python library for visualization purposes and does not involve public map data sources.

can cover a wide range of urban areas, from small neighborhoods to entire cities, and can span different time periods, from days to years. As stated before, the trajectories of humans or vehicles are constructed by combining the location data over time, creating a sequence of visited locations. To better explain, the mobile users' visited locations are plotted on a map using the Folium[1] Python library, as seen in Figure 4.3. Trajectory data can be used to extract a variety of mobility features, including travel patterns, activity hotspots, and trip purposes. Such information can be used to develop predictive models that can assist in the design of efficient and effective transportation systems.

Our proposed ANN-based mobility predictors' performance are evaluated on two mobility scenarios: a small-scale and a large-scale scenario, in comparison to a set of state-of-the-art mobility predictors. The two scenarios are named based on the size of the users' moving area. The large-scale scenario is based on the *Orange* telecommunication S.A. [104] private dataset, while the small-scale scenario is based on the *ETH* [68] plus *UCY* [47] public camera dataset.

### 4.4.1 Orange Dataset

In the *large-scale* scenario, we assume that users can move within an area that spans several kilometers. In this case, the information about the microscopic user mobility (small-scale coordinates) over a short time-interval is not relevant to characterize mutual interactions between users. Hence, we consider only the information about which base station the user is connected to. For the large-scale scenario, we use a private cellular network management dataset provided by Orange telecommunication S.A., France [104].

[1] https://python-visualization.github.io/folium/

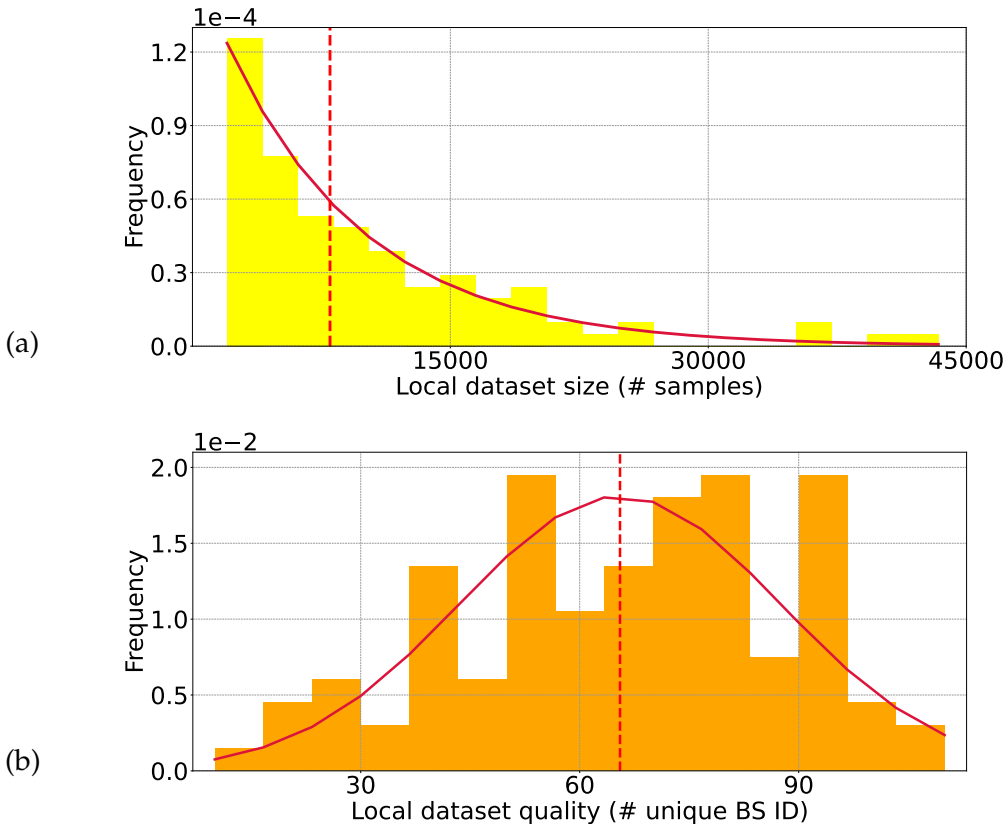



FIGURE 4.4: Distribution of the dataset size (a) and quality (b) for 100 random users in the large-scale scenario (Orange dataset) [18] ©2023 IEEE.

This dataset contains the timestamps and the cellular connections of 1.3 million users that move near a district of Paris between July and September 2019. For each user, this dataset holds information regarding user Global eNB id as the anonymized unique ID of a connecting base station, the start and end time of the user connected to a base station, and anonymized IMSI as the user ID. Note that we pick the connections' starting time as user timestamps. For privacy reasons, the exact location coordinates of the subset of 131 identified base stations are inaccessible, and the user identities are anonymized. Trajectories in the large-scale scenario will contain sequences of a few thousand information vectors $p_u = (t_u, b_u)$, sequence of timestamps and base station IDs as mentioned in Section 4.2.1, where the timestamps of any two consecutive vectors are separated by a few minutes from each other. We assume that the prediction models deployed in the large-scale scenario observe each user's trajectory for the past $T_{\text{obs}} = 16$ timestamps and predicts the trajectory for the future $T_{\text{pred}} = 1$ timestamp.

To reduce simulation time, and without loss of generality, we assume that this scenario contains 100 users randomly sampled from the Orange dataset. We generate a local dataset for each user by splitting the Orange dataset into several sub-datasets, grouping samples by unique user

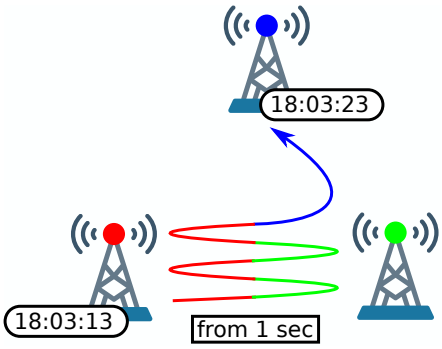



FIGURE 4.5: A user's several ping-pong handovers between surrounding BSs within a minute [104] ©2022 IEEE.

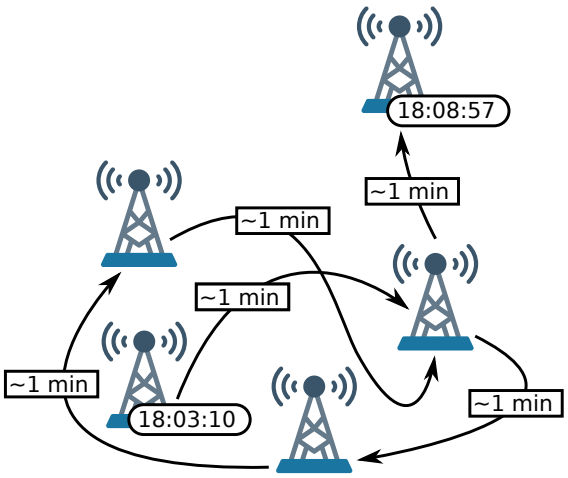



FIGURE 4.6: A user's consecutive connections to the same BS and cycle of connections [104] ©2022 IEEE.

IDs, and storing these local datasets on each user to simulate both centralized- and distributed-dataset scenarios for isolated and interactive mobility predictions. In Section 4.4.1, we provide a detailed description of the preprocessing steps applied to the Chaos Orange dataset and the selection of individual user data after cleaning. Figure 4.4a shows the distribution of 100 random users' dataset sizes expressed in number of data samples. The bars represent the frequency histogram of each user's dataset size expressed in its number of samples. The red curve represents the pdf of an exponential distribution $\mathcal{E}(\lambda)$ fitted on the data, with $\lambda = 1/8088.59$. Figure 4.4b shows the distribution of 100 users' dataset quality. High-quality datasets contain samples over a larger set of unique visited Base Station IDs. The bars represent the frequency histogram of each user's number of unique base stations visited. The red curve represents the pdf of a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ fitted on the data, with $\mu = 65.52$ and $\sigma = 22.02$.)

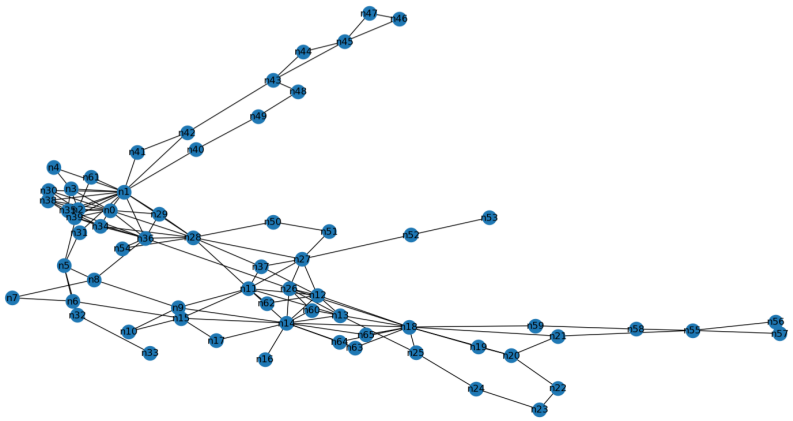


FIGURE 4.7: Re-constructed topology of an user's connected base stations [104]


**Orange Data Preprocessing**

The main challenges regarding Orange data preparation are the immense data size and to discriminate characteristics of the dataset. It is important to note that we received exclusive access to this dataset from Orange S.A. and are the first researchers to work with it and perform preprocessing. 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 600GB, including information of more than a million users, which are not homogeneous in terms of the data sampling rate, as shown in Figure 4.4a. 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 base stations. However, short connection times (a few seconds) with base stations might not be caused by user mobility. This behavior could be because users connect to more than one of the surrounding base stations or perform ping-pong handovers between neighbor base stations due to varying received signal strength. In this work, we consider a ping-pong handover, according to the definition of Tartarini *et.al.* [83], as a disconnection and re-connection to base stations within 4 seconds.

Figure 4.5 illustrates how users suffer from ping-pong handovers among surrounding base stations within the same minute. Figure 4.6 shows how sometimes users hop back to the first base stations within a short time period, forming a cycle of connections.

The reason behind the frequent ping-pong handovers and bounces in the Orange dataset is unknown to us due to the lack of precise locations of users and base stations, signal power, speeds, and direction of users' movements. Since the Orange dataset does not provide location and GPS coordinates of base stations, to visualize the base stations' distribution, we extract the relative topology from the mobility traces to re-construct the relative base stations' topology. From each user's trace data, we detect unique visited base stations, and then for each unique base station, we extract all consecutive connections and assume them as the base station's neighbor nodes. Finally,

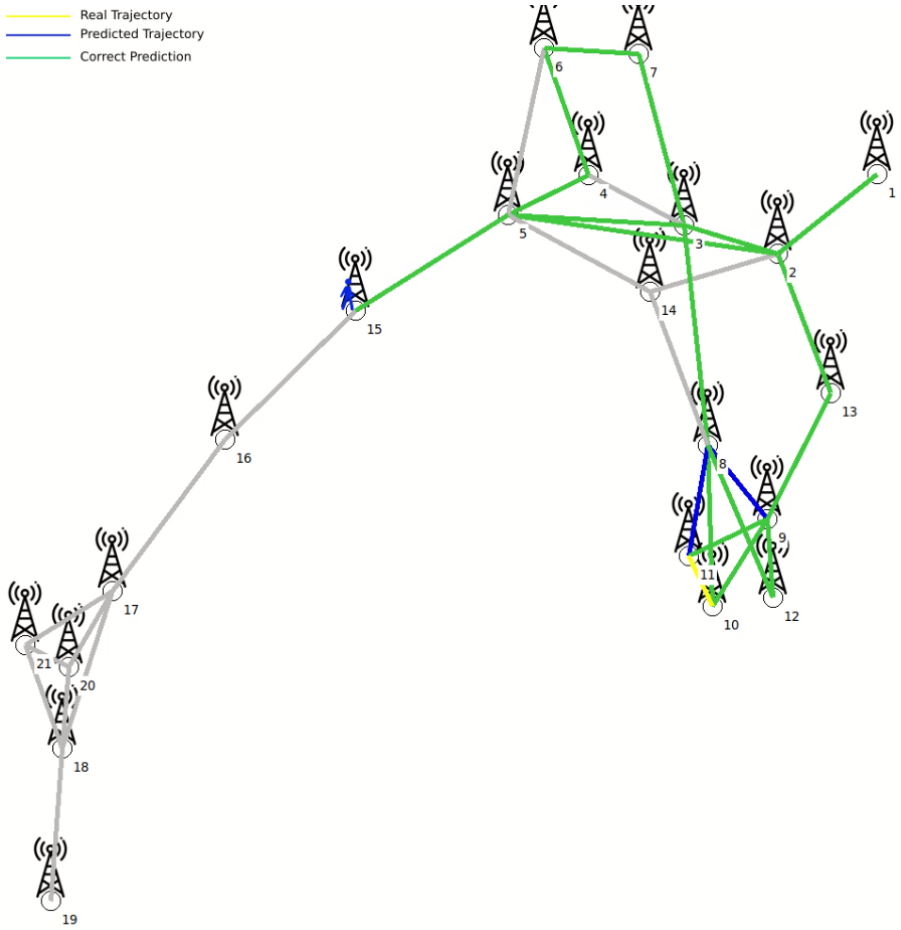



FIGURE 4.8: Demo of single-user trajectory prediction across cellular base stations on the reconstructed topology of Orange dataset.

we assign relative coordinates to all the nodes (base stations). Figure 4.7 shows an example of a re-constructed topology from the trace data of a random user within 30 days using the NetworkX[2] Python library. During this period, the user has moved among 101 different base stations (out of a total of 131 base stations). Figure 4.8 and Figure 4.9 display screenshots from our developed animation of single- and multiple-user mobility prediction across 19 and 30 cellular base stations, respectively, on the reconstructed topology of the Orange dataset. This animation serves as a demonstration of our proposed trajectory predictor, which was presented in the Demo section of the Orange project.

[2]https://networkx.org/

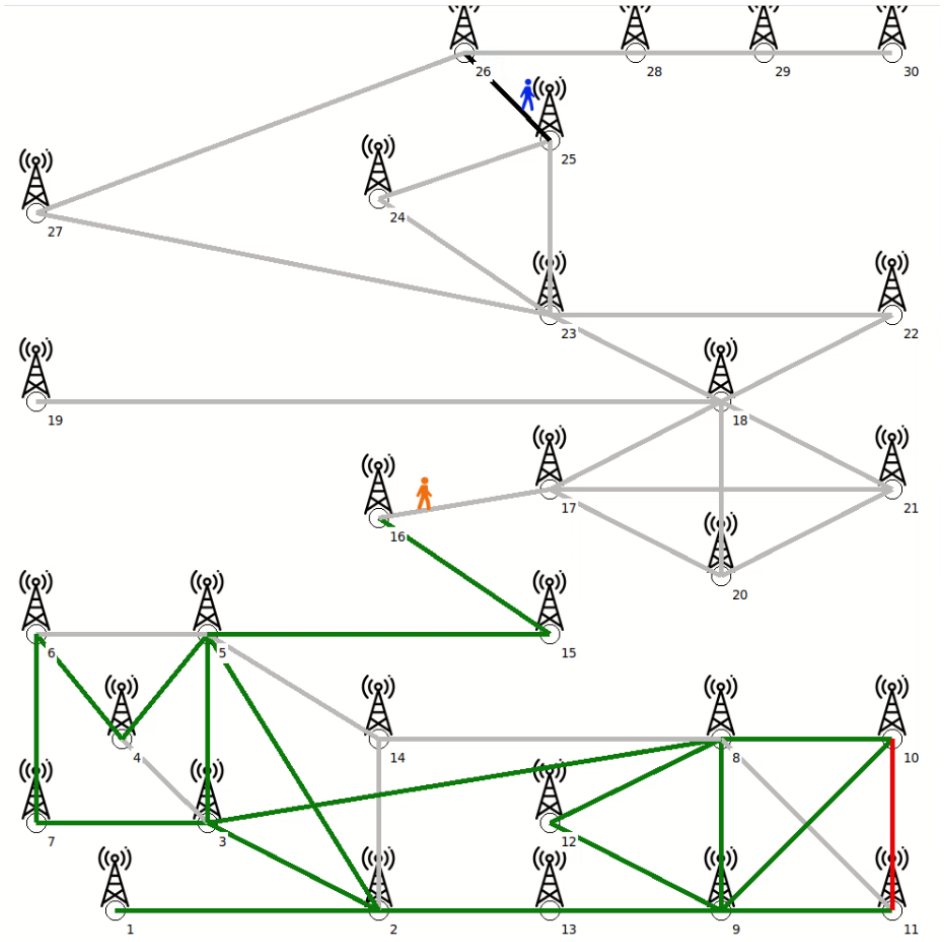



FIGURE 4.9: Demo of multiple-user trajectory prediction across cellular base stations on the reconstructed topology of Orange dataset.

### 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. As shown in Figure 4.4a, some users possess a large number of data samples, while others have a small number. Additional information would be required for users with insufficient mobility data, indicated by falling on the right side of Figure 4.4a). Social data, such as users' professions, ages, education levels, job locations, and attended social events, may provide valuable context to enhance the quality of their mobility data. However, this type of information is currently unavailable in the Orange dataset. Therefore, we cluster users based on their number of data samples into two classes of *good quality* 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 of 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 base stations in consecutive timestamps. In such cases, we keep the first unique sample (base stations) and remove redundant ones. Therefore, the threshold on the number of data samples is chosen in such a way that after cleaning duplicated successive base stations, 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 [30].

#### 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 base stations 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 proposed to evaluate for each user its time and frequency domain *regularity ratio* metric as described in Section 4.6.

### 4.4.2 ETH+UCY Dataset

In the *small-scale* scenario, we assume that users move within an area that spans a few tens of meters. Therefore, the information about the base station to which users are connected does not characterize inter-user mobility interaction, nor contribute to improving the mobility prediction task. For this reason, we disregard any information about base stations and only consider small-scale user position coordinates in their trajectories. For the small-scale scenario, we use the information contained in the ETH [68] and UCY [47] public datasets[3] containing video streams of real-world, small-scale pedestrian mobility in urban scenarios captured from bird-eye-view cameras, where users interact with each other and influence respective movements according to real social interactions at a microscopic scale. An example of a mobility scenario captured from a bird's-eye-view camera can be seen in Figure 4.10.

[3]For more information regarding ETH dataset check: https://paperswithcode.com/dataset/eth, and regarding UCY dataset check: https://paperswithcode.com/dataset/ucy.


FIGURE 4.10: An example of the ETH dataset mobility scenario. This image is sourced from the *Papers With Code* website https://paperswithcode.com/dataset/eth which initially is sourced from the *Medium* website https://medium.com/@zhenqinghu/pedestrian-detection-on-eth-data-set-with-faster-r-cnn-19d0a906f1d3. The *Papers With Code* website is a free resource with all data licensed under CC-BY-SA, which is different than the license of this thesis. See https://creativecommons.org/licenses/by-sa/4.0/deed.en.

From both video datasets we extract the trajectories of 1536 pedestrians from five different urban scenarios with sampling rate of 0.4 s following the image processing method proposed at [28], and save unique users' data separately to simulate both centralized and distributed ML training scenarios. The ETH dataset contains two urban scenarios named ETH and Hotel whereas, the UCY dataset contains three scenarios named Univ, Zara1, and Zara2. As a result, each trajectory $T_u$ in the small-scale scenario will contain a sequence of up to 100 user information vectors $p_u = (t_u, x_u, y_u)$, where the timestamp granularity of any two consecutive vectors is 0.4 s. We assume that the prediction models deployed in the small-scale scenario observe each user's trajectory for the past $T_{\text{obs}} = 8$ timestamps (3.2 s) and predicts the trajectory for the future $T_{\text{pred}} = 12$ frames timestamps (4.8 s). To ensure a fair comparison with other state-of-the-art trajectory predictors that use the same dataset, we selected the values of $T_{\text{obs}}$ and $T_{\text{pred}}$ from the literature.

## 4.5 Data Preparation and Feature Extraction

Input data preparation for time-series trajectory prediction is a crucial step as raw data is often not in a convenient structure for researchers and lacks organization for ML applications. Data

preprocessing plays a vital role in transforming the data and creating relevant features that are useful for accurate prediction. Proper feature extraction is fundamental in applying supervised machine learning algorithms.

Features can be categorized into various groups, including temporal and spatial features. Temporal features capture context information related to the duration of a visit. Users tend to exhibit certain temporal patterns when visiting specific places, such as regular office hours or lunch breaks. These temporal patterns can provide valuable insights for prediction models.

Spatial features focus on contextual information related to the geographical aspects of visits. They aim to measure aspects such as the frequency of a user's visits to a particular location and the pattern of periodicity and sequence of visited locations over time. Combining location and time information provides a comprehensive understanding of the trajectory patterns and enables the modeling of location dynamics.

In addition to these features, cosine and sine transformations of location coordinates can be utilized to capture cyclic patterns and improve predictions for periodic behaviors. By incorporating these transformed coordinates, the model can better capture the cyclic nature of certain activities or movements that exhibit regularity, such as daily routines or seasonal patterns.

Feature extraction in time series prediction is a crucial step that involves transforming raw time series data into a set of representative features, capturing essential information for the prediction task. In the context of our approach, the input to the ANN consists of a *multivariate time series*. This time series encompasses multiple sequences of time stamps, visited location IDs and GPS coordinates, as well as cyclic cosine and sine features derived from the location sequence.

By incorporating both temporal and spatial features, as well as combinations of location and time, along with cosine and sine transformations, in the data preprocessing stage, important contextual information can be effectively captured. This comprehensive approach enhances the accuracy of time series trajectory prediction models, allowing for more precise and meaningful predictions in various applications such as transportation planning, location-based services, and urban mobility analysis.

## 4.6 Data Quality and Periodicity Estimation

As presented in Section 1.3.4, we propose *regularity ratio* metrics in both the time and frequency domains as a measure of data quality and periodicity to identify users whose data is easier to learn for an ANN. We compute the regularity ratios for each mobile user, and utilize the resulting values to select only eligible users for training tasks. By doing so, we aim to reduce computational complexity and costs. The pretrained models are then migrated to the remaining silent users. If both regularity ratios $\rho_j^t$ and $\rho_j^f$ meet a predefined threshold that we have set for data quality and periodicity, then the users are deemed eligible to be included in the system.

The acceptable threshold on ratios for peaking users as eligible users is determined empirically based on the characteristics of the available dataset. This allows us to distinguish between users whose data is likely to be learned easily by an ANN and those whose data may be more difficult to learn. By selecting users with higher-quality data, we can improve the overall performance and accuracy of the ANN-based system.

TABLE 4.1: LSTM architectures suggested by RL agent for heterogeneous and homogeneous user data [104] ©2022 IEEE

| user ID | total visited BSs | neuron/LSTM layer | neuron/dense layer | no. layers |
|---|---|---|---|---|
| 59 | 110 | 16 | 150-40-110 | 3 layers |
| 16 | 105 | 16 | 80-150-102 | 3 layers |
| 10 | 49 | 16 | 150-49 | 2 layers |
| 81 | 22 | 16 | 40-20 | 2 layers |

### 4.6.1 Time Domain Signal Processing

According to our experiments, the complexity of each suggested NN architecture by an RL agent is affected by the type of user movement. Users with large numbers of visited base stations (heterogeneous movement) and less periodic behavioral history have been suggested architectures with deeper hidden layers. In contrast, users with a few numbers of visited base stations (homogeneous movement) and predictable movements got simpler architectures.

Table 4.1 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. An observable trend is that as the number of visited locations in a user's history log increases, more complex and deeper ANNs are required to accurately predict trajectories using RL techniques. This suggests that the increased complexity and depth of ANNs are necessary to capture and learn the intricate patterns and dependencies present in the trajectory data as the user's history becomes more diverse and extensive.

Let us define $|T_j|$ as the *length* of the trajectory $T_j$. For datasets containing mobility information as the sequence of Base Station IDs to which each user connects, we define $D_j$ as the number of *unique visited base stations* that appear in the trajectory $T_j$. Figure 4.1 shows the user trajectory, between points A and B in an area, defined as the sequence of visited base stations. As stated in Section 4.2.1, in such a scenario, trajectory prediction aims to predict the next base station ID which is a classification task. As is shown in Figure 4.1, we can count the number of unique visited base stations to compute $D_j$ in classification problems. In contrast, for datasets containing mobility information as geographical GPS location coordinates, we define $D_j$ as the number of *unique visited areas* of the trajectory $T_j$. In this case, each location coordinate is mapped to an area through spatial indexing. Figure 4.2 shows the user trajectory, between points A and B in an area, defined as the sequence of GPS latitude-longitude coordinates. As stated in Section 4.2.2, in such a scenario, the aim of trajectory prediction is to predict the next GPS coordinate which is a regression task. In this case, we first divide the area into a grid and group multiple coordinates into smaller areas, assigning each area a unique index. Then, we can calculate the number of distinct areas visited and use this to compute $D_j$ in regression problems. This approach facilitates the detection of periodicity in the visited areas of datasets that contain numerical coordinates, as depicted in Figure 4.2.

We define the time domain regularity ratio for the $j$-th user as $\rho_j^t = |T_j|/D_j \in [1, +\infty)$, which is proportional to the user trajectory quality. We design such a trajectory quality metric after

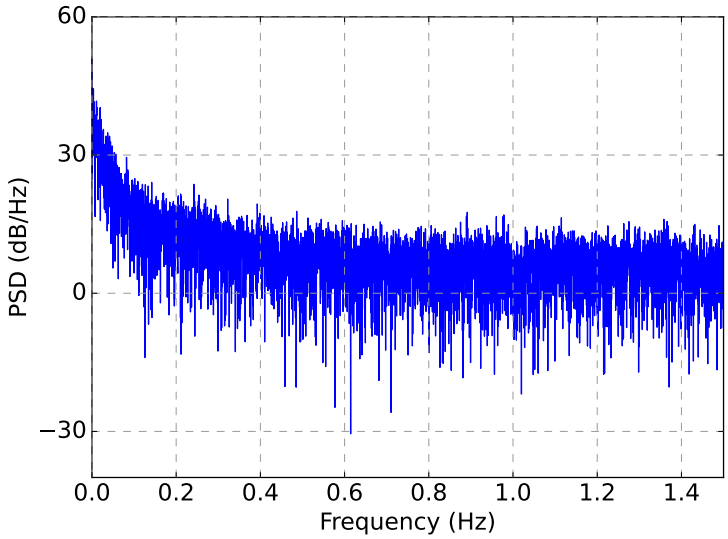



FIGURE 4.11: Trajectory data signal in frequency domain.

observing that users who produce more data samples while visiting a limited number of locations display more regular mobility patterns and reach higher prediction accuracy as shown in 5.5.

### 4.6.2 Frequency Domain Signal Processing

On the other hand, let us define $S_j$ af the *Signal-to-Noise (SNR)* Ratio and $F_j$ as the *dominant Frequency* of the trajectory $T_j$ in the frequency domain. we define the frequency domain regularity ratio for the $j$-th user as $\rho_j^f = S_j/F_j$. The dominant frequency in a signal refers to the frequency component with the highest amplitude in its power spectrum. If a signal is periodic, its power spectrum will have discrete peaks at the harmonics of the fundamental frequency, and the dominant frequency can correspond to the fundamental frequency of the periodic signal. Over the course of experiments, we observed that users with relatively high dominant frequencies and high signal SNR tended to exhibit strong oscillations in their sequential data and frequently visited multiple locations. This behavior made it easier for a neural network to detect periodic patterns in their data. On the other hand, users with extremely low or high dominant frequencies may have visited a wide range of locations or not moved significantly, resulting in non-periodic behavior that was more difficult for the neural network to detect.

Similar to the time domain, the SNR and the number of data samples can be indicative of data quality in the frequency domain. In addition, the dominant frequency and number of unique visited base stations are related and similarly informative for showing the periodicity of user movement. Users with relatively low (but not very low) numbers of unique visited locations and relatively high (but not very high) periods of oscillation in their data exhibit a clear periodic pattern that can be effectively predicted by neural networks.

Figure 4.11 shows a random mobile user's trajectory signal data in the frequency domain from the Orange dataset. To process such a signal we divide it into smaller portions to possibly extract the periodicity, calculate its highest peak and dominant frequency, and the power of signal and noise. Furthermore, we apply smooth algorithms to better interpret the signal.

## 4.7 Evaluation Methodology

In this study, we compare the performance of our four proposed trajectory predictors, named, RC-TL, INTRAFORCE, FedForce, and GTP-Force, against several non-NN- and NN-based mobility prediction models. Our evaluation is conducted on both large-scale and small-scale scenarios, using the Orange and ETH+UCY datasets, as presented in Section 4.4.1 and Section 4.4.2, respectively. Following this, we will describe our system setup and the evaluation metrics used to assess its performance.

### 4.7.1 Mobility Prediction Experimental Setup

We have utilized several Python open-source libraries to implement various machine learning, neural network predictors, reinforcement learning processes, federated learning, and game theory simulations. Specifically, Keras[4], TensorFlow GPU, TensorFlow Federated, NetworkX, PyTorch, Scikit-learn, Matplotlib, Seaborn, Numpy, Pandas, and Nashpy libraries were employed in our implementation.

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 [5]) on an eight-core machine of Intel(R) Xeon (R) E5-2630 v2 @ 2.60 GHz with 4 GB RAM process environment. The UBELIX cluster presently comprises approximately 320 compute nodes, boasting roughly 12,000 CPU cores and 160 GPUs, amounting to nearly one million GPU cores. Our building models use a single core (both CPU and GPU cores) for each user on UBELIX enabling parallel execution of the predictor for multiple users. Note that the implementation does not depend on a specific set of hardware but only needs robust software library configurations.

In Chapter 8, we model the average wireless channel throughput $B_t$ between local mobile clients and the server at each time slot $t$ as a uniform distribution from 10 Mbit/s to 100 Mbit/s, formally $B_t = \mathcal{U}([10, 100]), \forall t \in \mathbb{N}$. We categorize the throughput values into three intervals: low throughput, from 10 Mbit/s to 40 Mbit/s, medium throughput, from 40 Mbit/s to 70 Mbit/s, and high throughput, from 70 Mbit/s to 100 Mbit/s.

TABLE 4.2: Mobility Management Simulation Parameters [104] ©2022 IEEE

| 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 |

TABLE 4.3: Service Migration Simulation Parameters [104] ©2022 IEEE

| 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 |

### 4.7.2 Mobility Management Experimental Setup

#### Hanover Management

We implemented the Reinforcement Learning-based Handover for Edge Computing (RL-HEC) algorithm in the NS-3.30[6] 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 4.2. 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 one device generating a constant bit rate traffic of 1 Mbps.

#### Service Migration

We implemented the Reinforcement Learning-based Service Migration (RL-SM) algorithm as well in NS-3.30 network Simulator. Table 4.3 shows the main parameters used in the simulations and

[4]https://keras.io/
[5]https://docs.id.unibe.ch/ubelix
[6]https://nsnam.org

results are shown with a confidence interval of 95%. 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. [44]. 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. Simulations are conducted 33 times, with different random seeds and users set from the dataset present in the scenario.

### 4.7.3 Mobility Prediction Evaluation Metrics

We define a set of metrics to evaluate the performance of our proposed mobility predictors compared to other state-of-the-art predictors in both large- and small-scale mobility scenarios.

#### Large Scale Scenario

To measure the model performance for the classification task of predicting future base station IDs for each user, we define *accuracy* as the ratio between correctly predicted next locations and the total number of predictions made by the model. Formulating the large-scale scenario through classification instead of regression (predicting the next ID and not the exact location coordinates) is due to the unavailability of the base stations' coordinates in the large-scale Orange dataset.

To measure the computational performance, we measure *build time*, *train time*, *model size*, and *computational complexity* and *resource consumption* defined as the time to build and train the personalized model by RL agent, the average time to train the RL-designed model by all systems users, number of RL-designed NN model's training parameters, and the total amount of computational resources spent in developing individual-social, centralized-distributed, or cooperative-non-cooperative trajectory prediction models.

To measure the communication performance, we evaluate *transmission time* and *communication overhead* defined as the transmission time of an RL-designed mobility predictor model to the server and the total communication volume to migrate the trained global model from distributed FL participants to users who were not allowed to participate in training.

#### Small Scale Scenario

To measure the model's performance in the regression task of predicting future location coordinates for each user, we define the *displacement error* as the average squared Euclidean distance ($E$) between the predicted points of a user's trajectory ($\hat{x}_u^t$, $\hat{y}_u^t$) and the true locations ($x_u^t$, $y_u^t$) over a *prediction window* of $T_{\text{pred}}$ future time stamps, given the user's history of visited locations over an *observation window* of $T_{\text{obs}}$ past time stamps as:

$$E(u) = \frac{1}{T_{\text{pred}}} \sum_{t=T_{\text{obs}}+1}^{T_{\text{obs}}+T_{\text{pred}}+1} (\hat{x}_u^t - x_u^t)^2 + (\hat{y}_u^t - y_u^t)^2. \tag{4.1}$$

We evaluate the model's trajectory prediction performance using the A*verage Displacement Error* (or *Mean Square Error*) as:

$$ADE = \frac{1}{n} \sum_{u=1}^{n} E(u), \tag{4.2}$$

which is averaged over the $n$ users.

### 4.7.4 Mobility Management Evaluation Metrics

We define a set of metrics to evaluate the performance of our proposed mobility prediction-based handover management and service migration algorithms compared to other state-of-the-art similar works.

#### Handover Management

To measure the proposed proactive handover management algorithm's performance, we define the number of *ping-pong handovers* and the average *throughput*.

Ping-pong handovers refer to the count of how many times a mobile device switches back and forth between two different base stations or access points within a certain period, which can indicate potential problems with network coverage, signal strength, or interference. High ping-pong handovers can lead to a degraded user experience, as the device may experience intermittent connectivity or decreased network performance.

Average throughput measures the average amount of data that can be transmitted over a network in a given time period. It is used to evaluate the quality of the connection between a mobile device and a base station or access point. Higher average throughput values indicate a faster and more reliable connection, while lower values may indicate network congestion or other issues that can affect the user experience.

#### Service Migration

To measure the proposed anticipatory service migration algorithm's performance, we define *latency*, number of *migration attempts*, and number of *migration failures*.

Latency is the time delay between when a user requests a service and when the service is delivered. For the proposed anticipatory service migration algorithm, latency refers to the time it takes for the migrated service to become available to the user.

The number of migration attempts represents how many times a service migration is attempted from one location to another. Each migration attempt consumes network resources and can potentially cause service disruption or increased latency.

The number of migration failures refers to instances where a service migration attempt is unsuccessful, which can occur due to network congestion, connectivity issues, or other technical problems. Migration failures can lead to service disruptions, increased latency, or other negative impacts on the user experience. Therefore, it is essential to monitor migration failures and minimize their occurrence.

## 4.8 Chapter Conclusions

In conclusion, this chapter has provided a strong foundation for the development of accurate and reliable trajectory predictors by offering a comprehensive understanding of mobility and trajectory prediction, including the creation of a mobility scenario to simulate the TP process, a discussion of mobility datasets, and techniques employed for data preprocessing and data periodicity estimation. Additionally, the chapter has introduced the evaluation metrics used to assess the performance of mobility predictors and mobility-relying wireless services. Building on this knowledge, the upcoming chapters will focus on the development of our four novel trajectory predictors and their performance evaluation, which are critical for improving the performance and reliability of wireless network services such as handover management and service migration.

# Chapter 5

# Reliable and Computationally-Light Trajectory Prediction for Wireless Network Mobility Management

## 5.1 Chapter Introduction

In modern wireless networks, ensuring low latency communications and high QoS is crucial for delivering a satisfactory user experience. The handover procedure in mobility management plays a critical role in determining the performance of services consumed by mobile users. Furthermore, an efficient service migration scheme performed ahead of time can ensure service continuity for end-users. However, current mobility management approaches have limitations, either being reactive methods or naive predictive methods. To overcome these limitations, next-generation networks require alternative mobility management techniques that leverage robust learning and predictive behavior analysis of mobile users [65]. In this context, developing personalized neural networks using NAS methods for mobility and trajectory prediction based on individual users' movement features is critical. However, NAS mechanisms face challenges due to expensive computational costs for optimization and search. Therefore, exploring alternative optimization approaches is of great importance. In this chapter, we aim to address research questions stated in Section 1.2.1 as follows.

*"RQ 1.1: How can we achieve a higher level of computationally-efficiency and reliability in current NAS techniques for trajectory prediction, while maintaining a desirable balance between the accuracy of neural networks and their convergence rate?*".

*"RQ 1.2: How can we enable a resilient proactive handover mechanism that ensures high QoS and uninterrupted service continuity for network applications, such as service migration, by leveraging high-performance neural networks?*".

Our objective is to improve the performance of mobility management techniques by developing reliable and convergent mobility predictors. In pursuit of these goals, we introduce three

novel algorithms: the Reinforcement Learning-designed LSTM neural network (RL-LSTM) mobility predictor, the Reinforcement Learning-based Handover for Edge Computing (RL-HEC), and the Reinforcement Learning-based Service Migration (RL-SM) [104][1].

In response to RQ 1.1, neural architecture search plays a crucial role in designing neural network architectures that are tailored to specific user data. However, existing NAS methods, such as Auto-ML, often suffer from computational inefficiency as they rely on random or grid search methods to explore the performance of each neural architecture. Therefore, our objective is to find a more computationally-efficient NAS solution that improves the accuracy of neural network architectures while reducing the computational resources required for architecture search. Our first contribution lies in proposing RL-LSTM, a reinforcement learning-based deep learning method that designs a suitable LSTM architecture tailored to the unique mobility features of each user. Unlike grid search or Auto-ML models, RL-LSTM trains for a few epochs during each iteration, efficiently narrowing down the search space.

Moving on to RQ 1.2, the research problem to solve is how to address the integration of RL-LSTM into the handover process to facilitate proactive handover management, which ensures high QoS and uninterrupted service continuity for network applications, including service migration. Thus, our second contribution is the introduction of RL-HEC and RL-SM, proactive handover management and service migration approaches, respectively. These leverage the high-performance RL-LSTM mobility predictor to provide uninterrupted service for end-users. Through multiple experiments, we observe that the combination of mobility predictive analysis and proactive management strategies in our proposed techniques leads to a significant enhancement in the overall user experience in wireless networks.

The remaining sections of this chapter are organized as follows: In Section 5.2, we introduce our RL-LSTM approach for designing personalized trajectory predictors in urban environments. Section 5.3 presents the proposed proactive handover management RL-HEC, while Section 5.4 focuses on the RL-SM approach for service migration. Experimental evaluations and performance assessments are provided in Section Section 5.5. Finally, Section Section 5.6 concludes the chapter.

## 5.2 RL-LSTM Trajectory Prediction System Architecture

### 5.2.1 Reinforcement Learning for LSTM Architecture Design

As mentioned earlier in Section 2.4.1 and depicted in Figure 2.1, the architecture of the LSTM comprises multiple layers with various hyperparameters, including a number of hidden layers, LSTM layers, dense layers, dropout layers, and the number of memory cells or neurons per layer, dropout ratio, and the order of layers. In our LSTM-based mobility predictor design, we consider multiple layers of basic LSTM cell units to form a stacked LSTM network. The LSTM layers themselves consist of a chain of memory cells. These hyperparameters play a crucial role in the model's performance and its ability to capture spatio-temporal dependencies in mobility data. Selecting the appropriate hyperparameters is critical to achieving the desired level of accuracy and efficiency in the LSTM.

[1]Partially reproduced in this chapter – Copyright ©2011 IEEE.

To optimize LSTM hyperparameters and improve prediction accuracy while reducing training time, we employ an RL agent to search for high-performance networks tailored to the mobility characteristics of user data. We use value-based RL to automate and personalize the design of LSTM architectures for the task of trajectory forecasting.

As the performance of RL is stated in Section 2.6, we define a search space for the RL agent to explore. The search space consists of multiple LSTM NNs with different hyperparameter characteristics. The RL agent selects the optimal architecture from a finite and fixed space of admissible architectures, which we call the *state space*. The state space is a subset of all the possible combinations of values that the LSTM hyperparameters can assume. Specifically, the state-space dimensions are:

- The number of layers that make up the LSTM
- The type of each layer among LSTM, dense, and dropout layers
- The number of memory cells per each LSTM layer
- The type of activation function
- The number of perceptrons in each dense layer
- The dropout ratio of each dropout layer.

The RL-LSTM state space can become considerably large depending on the values of the mentioned parameters, making it impossible to test the performance of every LSTM architecture in the space through grid searching or even random searching. RL provides a method to search for the optimal architecture, avoiding an exhaustive grid search, through exploration and exploitation. As already stated, "exploration" and "exploitation" are two phases in RL. During the exploration phase, the RL agent tries out different actions to learn the rewards associated with them. In the exploitation phase, the agent exploits the knowledge it has acquired during the exploration phase to make optimal decisions. The goal is to balance exploration and exploitation to find the best strategy.

As shown in Figure 2.4, at the beginning of an episode, the agent takes an action $a$ from a subset $A(s)$ of the action space, where $A(s)$ depends on the currently observed state $s$. Every action adds one layer to the current LSTM and fixes the added layer's type and parameter values, leading the environment (i.e., the LSTM) into a new admissible state (i.e., architecture). In order to guarantee the arrival state is admissible, the new layer's type is constrained by a set of rules:

- First and last LSTM 's layers must be a LSTM and a dense layer, respectively
- The last dense layer's number of neurons must be compatible with the number of unique classes in the visited location history
- LSTM layers can be followed only by other LSTM and dense layers
- Dense layers can be followed by other LSTM, dense, and dropout layers

- Dropout layers can be followed only by dense layers.

After the agent has taken action, the corresponding reward is unknown and must be computed by training the resulting LSTM with the cluster's representative users' data for a limited amount of epochs (*exploration training*). The model's prediction accuracy on the mobility dataset is the reward associated with taking that action in that state. The RL agent uses the *Q-learning* algorithm with an *ε-greedy* strategy to learn the policy for selecting actions and stops searching the state space for better CNN architectures when it has reached either a *target accuracy* or a maximum number of episodes. The reward for taking each action in each state is computed using the Bellman equation, presented in Equation 2.18.

RL-LSTM sets $\varepsilon = 1$ at the beginning of the exploration phase as the probability of taking a random action. When the agent has taken a number of actions equal to the maximum number of layers allowed by the state space, it starts over from a state with a single layer and linearly decreases $\varepsilon$. The process is repeated until $\varepsilon$ reaches a minimum value $\varepsilon_0$ to complete the exploitation phase. In this way, the agent prefers randomly exploring the space during the first phase of the learning to identify promising architectures and gradually changes its policy to select actions that guarantee higher reward to identify higher-performing architectures.

Finally, RL-LSTM selects the LSTM architecture with the highest accuracy among all those explored by the RL agent and trains it, allowing a much larger number of epochs compared to the exploration epochs. We note that, during the search (exploration) phase, the RL algorithm generates and evaluates a large number of candidate LSTM architectures, and for each architecture, it trains the network for a small number of epochs to get a quick estimate of its performance. Once the search phase is complete and the best architecture is identified, the network is then trained for a longer period of time, typically hundreds of epochs, to achieve the best possible performance on the task at hand.

### 5.2.2 Transfer Learning for Expedition of the RL Process

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 transfer learning framework, which offers a way to transfer knowledge from the previously trained LSTM predictor to a newly suggested LSTM predictor with partially similar layers in order to speed up the searching process.

Transfer learning approaches can be applied both between different neural networks and within a single neural network from one layer to another. Transfer learning aims to leverage knowledge gained from one task or domain and apply it to another related task or domain, thereby improving the performance and efficiency of the learning process.

In this way, at each iteration, the newly suggested LSTM can pass the learning phase faster. In our RL-based NAS method, the process of adding a layer at each episode to form the new architecture follows a sequential model. When consecutive episodes result in LSTM architectures with similarities in terms of hidden layers, neurons per layer, and connectivity, we apply transfer learning to leverage this knowledge transfer. TL involves transferring the knowledge of similar layers from the LSTM architecture at iteration $t-1$ to the LSTM architecture at iteration $t$. The

knowledge here refers to the traiend weights of the LSTM NN architecture that is saved and transferred to the other partially-similar LSTM to be initialized with. By doing so, we can capitalize on the learned representations and architectural choices from previous iterations, enhancing the efficiency and performance of the NAS process.

Let the LSTM ardhitecture at episode $t-1$ contain $n$ layers denoted as: $L = \{l_1, l_2, \ldots, l_n\}$, 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 at episode $t$, 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: $\tilde{L} = \{\tilde{l}_1, \tilde{l}_2, \ldots, \tilde{l}_{n-1}, l_i', \tilde{l}_n\}$. We define the function $v(l_j) \in \mathbb{N} > 0$ 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: $L \bigcap \tilde{L} := \{l_i \mid v(l_i) = v(\tilde{l}_i)\}$ for $i = 1, \ldots, n-1$. Thus, we can transfer knowledge from $l_i$ to $\tilde{l}_i$ for $i = 1, \ldots, n-1$. Transferring knowledge means carrying and copying the first $n-1$ layers' neurons weights from the teacher to the student LSTM as: $\omega(\tilde{l}_i) := \omega(l_i), \forall i = 1, \ldots, n-1$.

## 5.3 RL-HEC Handover Management System Architecture

This section details our handover algorithm called RL-HEC, a service-aware handover algorithm based on a DL mobility prediction. RL-HEC takes advantage of the NAS-enabled high-performance individual trajectory prediction model trained for users to avoid ping-pong handovers 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 handover. This is achieved by assuming knowledge of the service instances' locations being consumed by users, since handover 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 the proposed RL-LSTM 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 handover as disconnection and reconnection to a certain Base Station (BS) within an interval of 4 seconds [83]. The prediction model for a given user can forecast with significant accuracy the next connection of a user and thus can be used to predict the occurrences of ping-pong handover and also service migration patterns.

We divide the handover procedure into three phases: *(i)* measurement, *(ii)* decision, and *(iii)* execution. In traditional signal-based handovers, 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 base stations' coverage areas, signal fluctuations often occur [1]. This causes unnecessary handovers, such as ping-pong handovers, 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 handovers by as much to almost zero, as shown in Section 5.5.3.

### 5.3.1 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 handover decision.

### 5.3.2 Decision Phase

The decision phase in RL-HEC happens within the handover Manager, which is located in the user's serving BS. The Handover Manager is a distributed entity running in every BS of the network. Therefore, each BS performs the handover 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 handover manager assesses the next handover predicted for the given user. In order to avoid ping-pong handover executions, the handover manager checks if the handover in question is a ping-pong handover. After a series of ping-pong handovers, 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 handover is actually executed, the handover algorithm must notify the RL-SM migration algorithm about the upcoming handover, so that any necessary service migrations can be performed in advance. This is possible, because handovers 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.

### 5.3.3 Execution Phase

We consider that a sudden handover 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 handover must be executed as soon as possible, and a discontinuity for the service being consumed may be inevitable.

**Algorithm 1:** RL-HEC Algorithm [104] ©2022 IEEE

```
   Data: Prediction depth
 1 for each connected UE do
 2     if UE has prediction model then
 3         if Prediction accuracy is above threshold then
 4             Perform mobility prediction;
 5             if Handover is expected then
 6                 if Next handoveris a ping-pong HO then
 7                     Define best BS as BS in which the client will stay the longest;
 8                 else
 9                     Define the best BS as the predicted BS
10                 Notify migration algorithm;
11                 while handovernot executed do
12                     Wait for service migration to finish;
13                     if Disconnection is imminent then
14                         Perform handover to best BS;
15                         Return handover status;
16                     Perform handover to best BS;
17                     Return handover status;
```

For users with a poor prediction model or without a RL-LSTM prediction model at all, a normal signal-based handover 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. 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 1. In this algorithm, RL-HEC checks for each connected User Equipment (UE), if a prediction model is available and, if so, assesses the prediction accuracy against a predetermined threshold. If the accuracy exceeds the threshold, mobility prediction is performed. The algorithm then determines if a handover is expected based on the predictions, selecting the best base station depending on the scenario. If it is a ping-pong handover, the best base station is defined as the one where the client is expected to stay the longest. Otherwise, the predicted base station is considered the best. The migration algorithm is notified, and the algorithm waits for ongoing service migration to complete. If disconnection becomes imminent, the handover to the best BS is executed. The handover status is returned, ensuring a seamless transition for the UE.

We consider that not all user models have the minimum accuracy in performing a good handover 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 handover avoidance mechanism can be accomplished by extrapolating the mobility

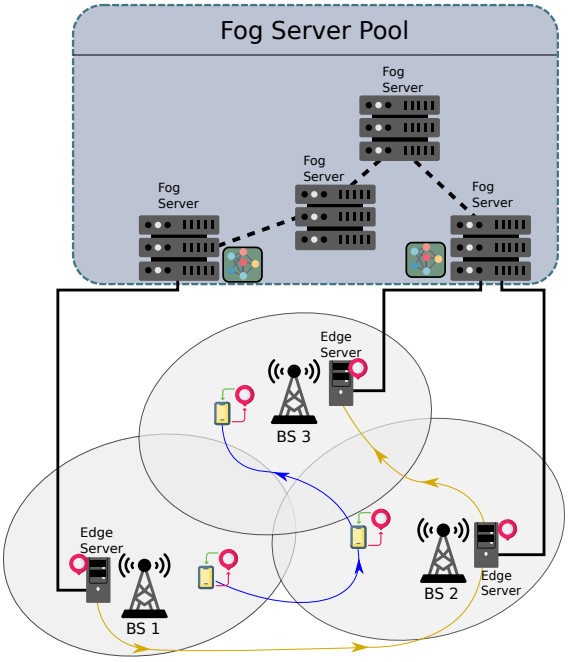



FIGURE 5.1: Multi MEC RL-SM Scenario [104] ©2022 IEEE.

prediction to the desired depth and checking if the predicted handovers are ping-pong handovers. After a handover 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 handovers. After that, a handover is scheduled.

## 5.4 RL-SM Service Migration System Architecture

This section discusses our proposed service migration strategy, socalled RL-SM, driven by our NAS-enabled mobility predictor. RL-SM relies on individual user RL-LSTM mobility prediction to perform proactive service migrations and guarantee service continuity. Furthermore, we consider the likelihood that UE will connect to a certain BS in the future to infer the best edge server to serve the user.

We assume an edge-enabled network such as shown in Figure 5.1. We see that users access the network through BSs with different coverage areas and follow some mobility patterns. 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.

RL-SM considers an edge-enabled network scenario, where users can consume services running on edge servers. As shown in Figure 5.1, users move and perform handover 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 [80]. 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 Equation 5.1, 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.

$$Q_t = \begin{cases} Q_s, & \text{if} \quad n = 1 \\ \alpha Q_s + (1 - \alpha) Q_{t-1}, & \text{if} \quad n > 1 \end{cases} \tag{5.1}$$

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 handover will be triggered for a given UE. This is achieved in conjunction with the handover algorithm operating in the network, which must report imminent handovers to the service migration framework. Each time the UE $u$ moves, there is the possibility that a handover will be triggered. Thus, the orchestrator predicts the user's next handovers. 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, RL-SM performs a lookup to find the edge servers associated with $u$'s next BS.

We divide the service migration procedure into two phases: *(i)* monitoring and *(ii)* assignment.

### 5.4.1 RL-SM Monitoring

The first decision of RL-SM 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 2 presents the RL-SM monitoring process. This algorithm focuses on making migration decisions for users in a network based on mobility predictions. It operates in a loop while the user remains connected. Firstly, mobility prediction is performed to anticipate the user's movement. The algorithm then estimates when handovers are likely to occur based on the predicted user trajectory. If a handover is expected in the near future, a migration decision is made. After the migration decision, the algorithm measures the quality of QoS experienced by the user. If the QoS falls below a predefined threshold, another migration decision is executed. This iterative process ensures that migration decisions are made based on mobility predictions and takes into account the QoS requirements of the user.

**Algorithm 2:** RL-SM Monitor [104] ©2022 IEEE

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

```
1 while user is connected do
2     Perform mobility prediction;
3     Estimate when handovers will be triggered based on the predicted user trajectory;
4     if handover is eminent then
5         Perform migration decision;
6     Measure QoS;
7     if QoS is below the threshold then
8         Perform migration decision;
```

### 5.4.2 RL-SM 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. RL-SM assumes that each edge server can assess the 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. RL-SM 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 RL-SM 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 3 shows RL-SM Assignment of an edge server to which an UE session shall be migrated.This algorithm's objective is to determine the best server for a user and perform migration operations based on certain criteria. The input consists of the user not being provided with the minimum requirements and a list of available servers. The algorithm aims to identify the ID of the best server for the user and conduct the necessary migration operations. It starts by listing the available servers and removing those that do not meet the resource requirements for the user's application. For each remaining available edge server, it retrieves the QoS for that server. The algorithm then enters a loop, continuously evaluating each server. It identifies the closest server to the user's future location, estimates the migration time, and checks if the migration can be completed before the user's arrival while ensuring that the server's QoS is above a specified threshold. If these conditions are met, the algorithm chooses the identified server as the target. In cases where the QoS of the current connected server falls below the threshold but the identified server's QoS exceeds it, the algorithm also selects the identified server as the target. If none of these conditions are met, the server is removed from the list. Finally, the algorithm performs the necessary migration operations. The overall performance of the algorithm lies in selecting the best server for the user based on proximity, estimated migration time, and QoS requirements, ultimately facilitating an optimal user experience.

**Algorithm 3:** RL-SM Assignment [104] ©2022 IEEE

**input :** User without 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;
```

The protocol for the execution of the migration algorithm is described in more detail in terms of the sequence diagram shown in Figure 5.2. 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.

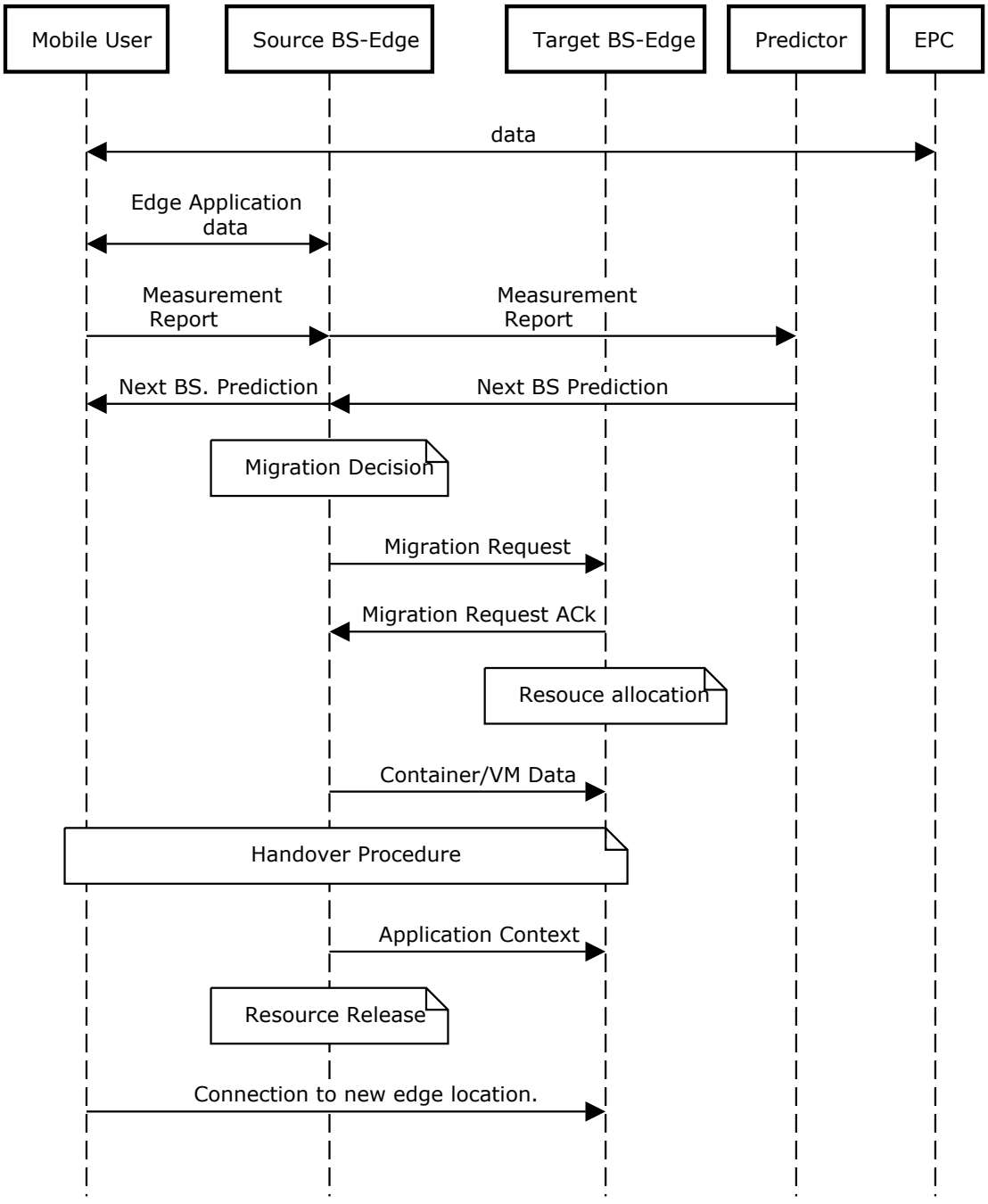



FIGURE 5.2: Sequence Diagram for Migration Procedure [104] ©2022 IEEE.

After the handover and a migration decision have been executed, the source edge server and the 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.

## 5.5 Evaluations

We evaluate the impact of location awareness through our proposed RL-LSTM mobility predictor on handover management and service migration performance in terms of QoS and QoE metrics, number of ping pong handovers, throughput, latency, number of migration attempts, and number of migration failures.

### 5.5.1 Experimental Details

In this thesis, we convert the time-series problem into a supervised learning problem, defining the input as multivariant one-dimensional arrays (i.e., user timestamps, trajectories, and cyclic features) and the output as the single sequence of visited base stations. The process through which an LSTM learns the associations between input and output is called *training*, and could be time-consuming. For this reason, to speed up the training process, we impose a limit on the number of training epochs and employ an *early stopping* training method to allow the training to end sooner if no significant accuracy improvement is made. In particular, we define $\Delta$ as the accuracy improvement threshold and the *patience* as the number of epochs that can elapse without an accuracy improvement higher than the threshold.

We set the batch size to 200 and the initial learning rate to 0.002. The Q-learning rate $\alpha$ and discount factor $\gamma$ are set to 0.01 and 1, respectively, to schedule the portion of immediate reward concerning the distant future reward of RL. We do not treat the Q-learning rate and discount factor as hyperparameters since the goal is to optimize the neural architecture.

During the RL exploration phase, we use 10-fold cross-validation to train and validate each user's data on suggested NN architectures for a few epochs.

Moreover, we split the data into a training set of 70% and a testing (or evaluation) set of 30%. 10-fold cross-validation is a technique used for training and validating a model on the training set of the data. Cross-validation allows all observations in the dataset to be trained and validated, and prevents overfitting. In this technique, the dataset is divided into ten equal parts or "folds". The model is then trained on nine of the folds and validated on the remaining fold. This process is repeated ten times, with each fold used as the validation set exactly once. The results of each validation are averaged to obtain a more reliable estimate of the model's performance. During this process, the model is trained on the training data (nine folds) and validated on the validation data (one fold).

The Q-learning agent's design process with epsilon-greedy policy involves choosing the probability of epsilon $\epsilon$, $Q$-learning rate $\alpha$, and discount factor $\gamma$ corresponding to Bellman's Equation. We set $\epsilon = 1.0$ in the initial episodes to ensure agent exploration and gradually reduce $\epsilon$ to 0.01 to move towards the exploitation phase. $\alpha \in (0, 1]$ determines the weight given to new information

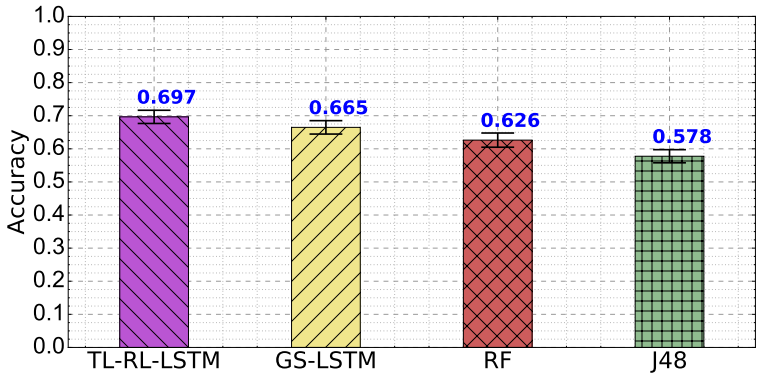



FIGURE 5.3: Average prediction accuracy of 100 users [104] ©2022 IEEE.

over old information, and $\gamma \in (0, 1]$ determines the importance given to immediate rewards over future rewards.

During each iteration $t \in 0, 1, 2, \ldots, 150$, the Q-learning agent in state $s \in S$ takes an action $a \subseteq A(s)$ and moves to the next state $s' \in S$. In each iteration, the agent generates a reward $r_t \in \mathbb{R}$ corresponding to the action-state, and the Q-table is updated. The state-action values are updated after each episode.

After the RL exploitation phase, we train the discovered LSTM for 200 epochs to evaluate the performance of the chosen architecture. We impose restrictions on the learning agent's actions in the action space. For instance, we terminate the predefined number of iterations earlier if the suggested LSTM achieves satisfactory prediction accuracy (e.g., 80%). Otherwise, we explore the entire defined search space. We also require the learning agent to have a dropout layer after each hidden layer to prevent overfitting, and we define a range of hyperparameters (e.g., number of hidden layers, number of neurons in each hidden layer, and dropout ratio) for the agent to try.

The LSTM search space to be searched by the RL agent is defined as 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 a 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 [62] of each NN dense layer.

### 5.5.2 RL-LSTM Evaluation Results

We evaluate the performance of the proposed RL-LSTM algorithm in this chapter. To illustrate the advantages of our proposed predictor, we compare the 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 GS-LSTM neural network. J48 and RF are non-NN predictors and thus, do not require neural architecture search schemes. 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 optimized architecture search method with respect to naive search approaches, yet it requires a remarkable training time to discover the best neural architecture. Thus, we implement a TL approach on top of the RL algorithm, proposing TL-RL-LSTM predictor, to further accelerate and optimize the searching process.

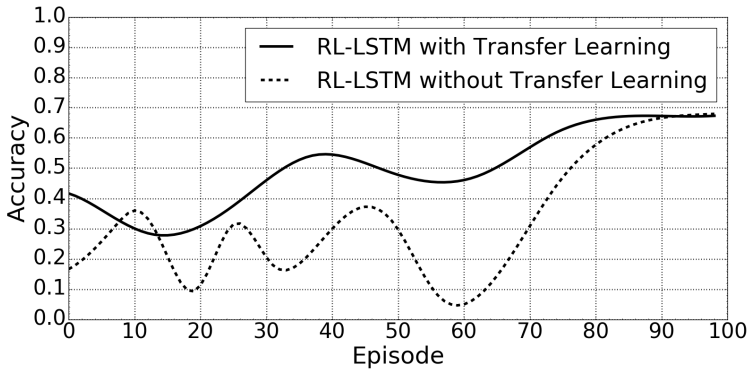



FIGURE 5.4: Average prediction accuracy of 100 users with/without TL [104] ©2022 IEEE.

Figure 5.3 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. This is because even though GS-LSTM is also able to find the optimal architecture, it takes a very long time to converge, so we define a smaller search space. Hence, there is no guarantee that the optimal architectures are included in the search space of GS-LSTM. 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.

Figure 5.4 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.

Transfer learning in neural architecture search involves the utilization of pre-trained networks to accelerate the training process. Specifically, by transferring $L-1$ similar layers from a pre-trained network to another network that shares $L-1$ similar layers, significant speed-up in training can be achieved. This technique not only enhances efficiency but also allows for the effective utilization of pre-existing resources, ultimately leading to improved performance and accelerated training in neural architecture search.

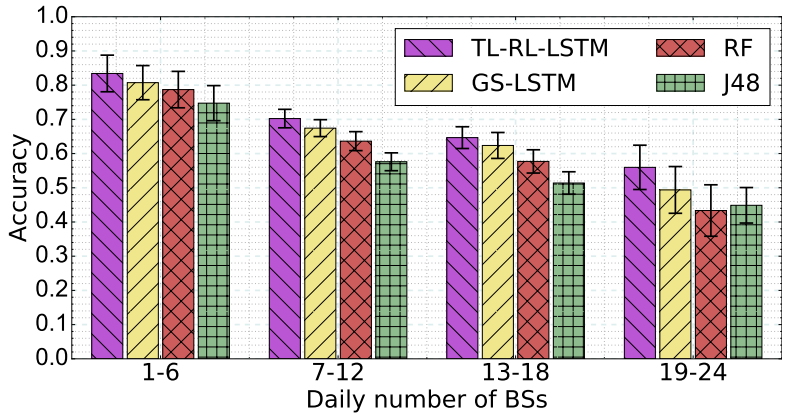



FIGURE 5.5: Different predictors' achieved accuracy grouped by the average number of unique daily visited BSs per User [104] ©2022 IEEE.

Over the course of the experiments, it has been observed that the accuracy achieved by each user depends on the number of visited base stations by each user. Users connecting to relatively fewer base stations tend to have more accurate prediction models. Figure 5.5 displays the accuracy of various predictors, organized by user mobilities. This visual representation provides compelling evidence in favor of the reliability of our proposed regularity ratio metric, as outlined in Section 4.6. Specifically, Figure 5.5 showcases the dependence between the frequency of visiting locations and the prediction accuracy, reaffirming the effectiveness of our approach in capturing and quantifying the relationship between user mobility patterns and prediction performance.

We define different levels of user mobility by estimating each user's average number of visited base stations per day. From right to left of Figure 5.5, 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 base stations, the high-mobility group contains 13-18 daily visited base stations, the medium-mobility group contains 7-12 daily visited base stations, and the low-mobility group contains 1-6 daily visited base stations. As it is shown, for each of the defined groups, RL-designed predictor maintains superior accuracy consistently. Further, we can see the trend of diminishing accuracy for very-high-mobility users. That is because as the number of daily visited base stations 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. The regularity ratio of very-high-mobility users group is quite a small value since the achieved accuracy is low and the mean number of daily base stations is relatively high. Alternatively, the regularity ratio of low-mobility users is quite a high value since the achieved accuracy is very high, and the mean number of daily base stations is low. The regularity 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.

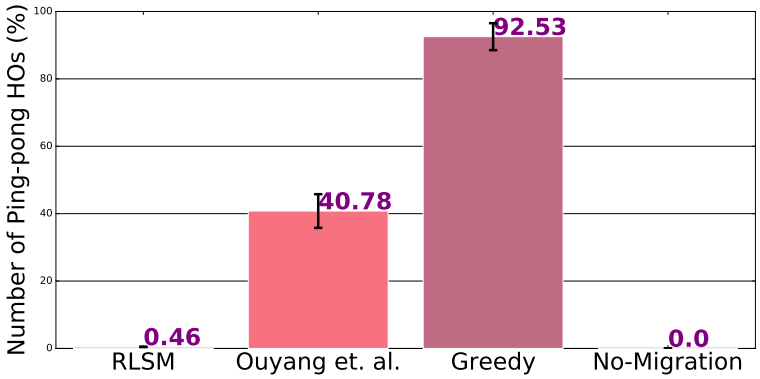



FIGURE 5.6: Percentage of Ping-Pong handovers for different algorithms [104] ©2022 IEEE.

### 5.5.3 RL-HEC Evaluation Results

We evaluate the performance of the proposed RL-HEC algorithm in this chapter. For comparison purposes, we implement three other state-of-the-art handover algorithms to test against RL-HEC. The algorithms are the following:
*(i)*: PRED [55] is position prediction-based handover algorithm based on the Reference Signal Received Power, Reference Signal Received Quality and some UE parameters like moving direction and the position inside the BS used as handoverdecision criteria.
*(ii)*: Received Signal Strength Indication (RSSI)-based [1], a standard handover algorithm based on signal events, meaning the events when the serving BS's signal quality drops below a threshold, 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 [1] uses a threshold value, *i.e.*, a handover 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 handovers and interference as well, but it can cause users to stay in overloaded macrocells.

Figure 5.6 shows the percentage of ping-pong handovers made in comparison with the total number of handovers 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 handover as a disconnection and reconnection to a BS within 4 seconds [83]. We can see that RL-HEC achieves a near-zero number of ping-pong handovers, 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 handovers, 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 handovers. The RSSI-based algorithm performs a much larger number of ping-pong handovers

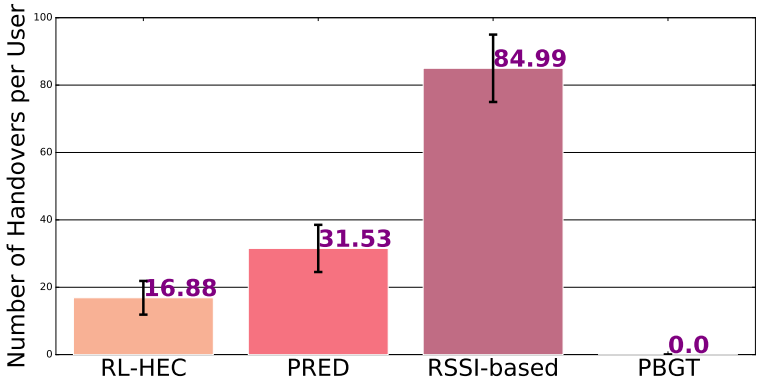



FIGURE 5.7: Number of handovers for different algorithms [104] ©2022 IEEE.

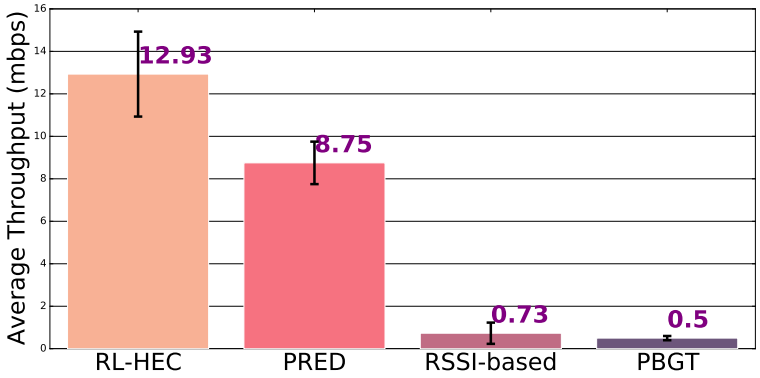



FIGURE 5.8: Average network throughput for different algorithms [104] ©2022 IEEE.

because it is more sensible to signal fluctuations, especially when the coverage area of neighbor BSs and the serving BS overlap. The Power Budget algorithm did not cause ping-pong handovers at all in the simulations. This is because it is executed with a significant hysteresis value that only allows a handover to happen, if the target BS's signals strength is a threshold above the serving one's.

The number of ping-pong handovers for each algorithm is proportional to the total number of handovers executed. Figure 5.7 shows the raw number of handovers with different algorithms. RL-HEC caused about 16 handovers 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 handover. Both RL-HEC and Power Budget performed less than 20 handovers per node. This is because in the decision phase of RL-HEC and Power Budget the UE often only performs a handover 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 Power Budget algorithm, connections are stable because most users are bound to the BSs with the highest transmission

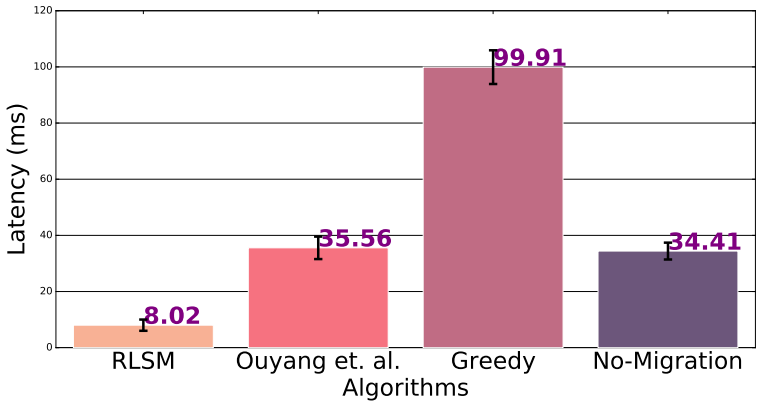



FIGURE 5.9: Average service latency for different algorithms [104] ©2022 IEEE.

power, ultimately increasing congestion levels and compromising QoS. This showcases how in scenarios with BSs of different transmission power value, traditional handover algorithms may fall short in performance. Each node under PRED performed on average one handover every 4 seconds. In the case of the RSSI-based algorithm, which is more sensitive to signal fluctuations, an excessive number of handovers is performed. In our experiments, with the RSSI-based algorithm, individual users perform one handover every 2 seconds.

Figure 5.8 shows the impact of each handover algorithm on 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 Power Budget 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 handovers. 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 handover executions.

### 5.5.4 RL-SM Evaluation Results

We now evaluate the performance of the RL-SM algorithm proposed in this work. To compare RL-SM with other works, we chose three other algorithms found in the literature:
*(i)*: Ouyang et al. [67] 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.

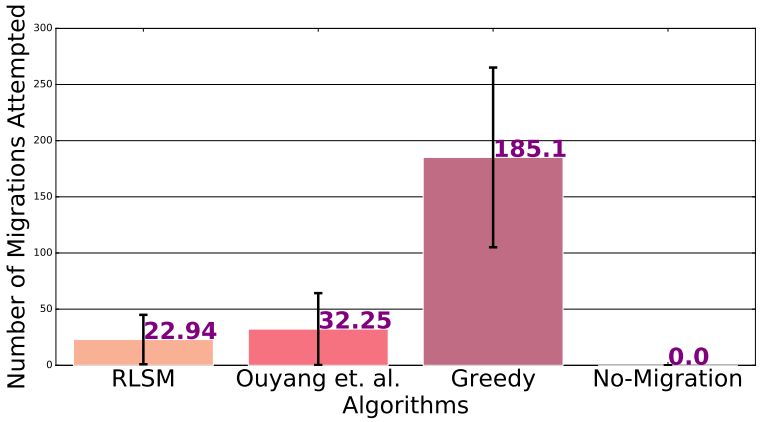



FIGURE 5.10: Number of service migration attempts for different algorithms [104] ©2022 IEEE.

*(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. *(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.

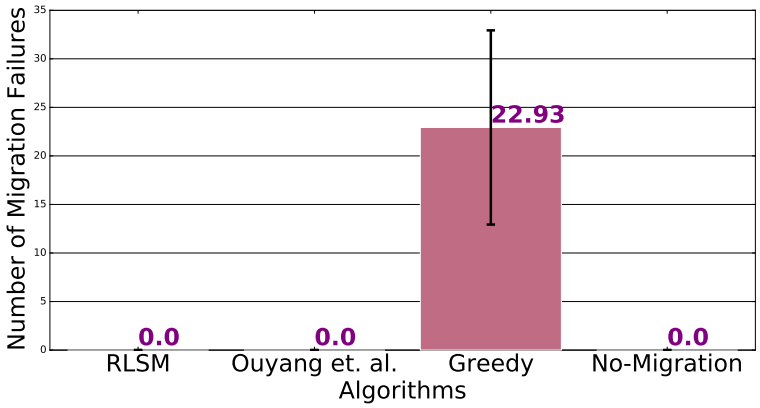



FIGURE 5.11: Number of service migration failures for different algorithms [104] ©2022 IEEE.

In Figure 5.9, we see the average latency across all users in the network for the service being tested. When considering the application's latency requirements, RL-SM 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 Figure 5.10. In the simulated scenario, RL-SM 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 the No-Migration algorithm, respectively. Since migrations in RL-SM 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. Figure 5.11 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. RL-SM 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 RL-SM.

## 5.6 Chapter Conclusions

User mobility awareness plays an essential role in enhancing network performance. In this chapter, we tackled the problem of how user mobility prediction can be used to deploy NN models to optimize network performance such as handover management and service migration. We designed an RL method to automate the architecture search for the LSTM mobility predictor with fast convergence rate using an inter-network transfer learning approach. We validated our ideas on a real-world large-scale anonymized dataset collected from the Orange telecommunication network operator. Experiment results show that our predictor delivers better accuracy over state-of-the-art mobility predictors. Moreover, we designed and implemented a novel handover algorithm RL-HEC and service migration RL-SM scheme, that benefit from our robust RL-LSTM mobility predictor. Simulation results show that the proposed solutions could reduce ping-pong handover 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 in general.

Our proposed RL-LSTM trajectory predictor exhibits remarkable improvements in prediction accuracy compared to other state-of-the-art neural networks that rely on heuristic network design or naive AutoML NAS methods. Additionally, it significantly enhances the performance of mobility management services. However, when considering large-scale networks, our RL-LSTM approach faces significant challenges in terms of scalability. In large-scale mobility networks, it becomes impractical to personalize the neural network architecture for each unique user. Therefore, in the next chapter (Chapter 6), we introduce a system designed to manage the complexities and computational demands associated with trajectory prediction on a large scale, while still maintaining high prediction accuracy. By addressing the scalability issue, our aim is to ensure that our trajectory prediction system remains effective and efficient, even in realistic scenarios involving large-scale wireless networks, handover exchanges, and service migrations for multiple users.

# Chapter 6

# Scalable and Convergent Individual-Agent Trajectory Prediction in Large-Scale Mobility Networks

## 6.1 Chapter Introduction

In the previous chapter, we presented a reinforcement learning-based solution that automates the design of neural networks for individual mobile users. However, when considering real-world wireless network scenarios with millions of user tags, the task of searching for an optimal neural network architecture for each individual user becomes computationally expensive in large-scale systems. To address this challenge, in this chapter, we develop scalable solutions for personalizing neural architectures in large-scale networks. By focusing on scalability, we can ensure that the process of personalizing neural architectures remains feasible and practical in real-world large-scale network environments.

In the existing literature, the majority of trajectory predictors are based on RNNs and their variants, particularly LSTM NNs. RNN models have been widely acknowledged for their high effectiveness in predicting mobility patterns through the analysis of historical location data and the learning of users' moving behaviors to forecast future locations. These models have demonstrated maturity in handling time series data. However, despite their successes, RNN-based trajectory predictors do have a few drawbacks. They tend to exhibit slower computational speeds and lack easy parallelization capabilities. Additionally, these models face challenges in retaining long-term memories of sequential time series data. These limitations indicate the need to explore alternative predictors and their performance in the field of trajectory prediction.

In this chapter, we aim to address research questions stated in Section 1.2.2 as follows.

*"RQ 2.1: How can we enhance the effectiveness of capturing complex spatiotemporal mobility features in trajectory prediction by substituting commonly used RNNs with other neural networks?*".

*"RQ 2.2: How can we scale NAS to personalize multiple individuals within large-scale networks? While scaling the computations, what is the optimal tradeoff between computational resource consumption and prediction accuracy that ensures efficient performance?*".

Our objective is to improve the performance of personalized individual-agent trajectory prediction in large-scale mobility networks. In pursuit of these goals, we introduce a novel model: Reinforcement Convolutional Transfer Learning (RC-TL) trajectory predictor [21][1].

In response to RQ 2.1, there is a need to explore alternative neural network architectures that can replace the commonly used RNNs in trajectory prediction. These alternatives should be capable of capturing complex spatiotemporal mobility features more efficiently. As a result, the first contribution of this chapter is the proposal of one-dimensional convolutional neural network-based trajectory predictors. Although CNNs are not commonly applied in learning time-series data, such as user mobility information, they offer great potential due to their ability to encode patterns in convolutional kernels. Additionally, CNNs support parallel learning and require minimal computation to perform the prediction task efficiently. To design the CNN architecture effectively, we leverage our previously proposed reinforcement learning model and incorporate it into the formation of RL-CNN predictors. This combination allows us to harness the strengths of both CNNs and reinforcement learning to create personalized trajectory predictors that can capture and analyze complex mobility patterns in a computationally efficient manner.

Moving on to RQ 2.2, even with the development of a more robust predictor RL-CNN, the challenge of scaling the NAS approach to personalize multiple individual users in large-scale networks persists. It is evident that any attempt to reduce computations and scale RL-designed neural networks will inevitably lead to a compromise in prediction accuracy. This is because reducing computational expenses often involves sacrificing certain computational neural layers or similar components. Consequently, the problem extends beyond finding a scaling method for personalization in large-scale networks, it also involves defining an optimal tradeoff between minimizing computational resource usage and maximizing prediction accuracy.

To address the computational burden of training, some studies have suggested clustering users with similar characteristics and training a single model for all users within a cluster. This approach has proven effective in reducing computational requirements in trajectory prediction tasks. However, no previous studies have explored the potential of user clustering specifically for reducing computational requirements in NAS. This objective serves as the second contribution of this chapter. In this direction, we propose the clustering of similar-trajectory mobile users, followed by the training of a single RL-CNN predictor per cluster. By sharing this predictor among users with similar behavior, we aim to significantly reduce the computational overhead associated with building the RC-TL system. This approach allows us to strike a balance between computational efficiency and prediction accuracy, thus overcoming the limitations of scaling NAS for personalized trajectory prediction in large-scale networks.

To increase the average accuracy across various users, personalizing NN tailored to different user datasets can be an adequate approach. While this provides sufficient personalization for each user to consider different statistical features, it cannot be scaled for a large number of users. Inspired by these shortcomings, we design the RC-TL system taking into account clusters of users with similar mobility features to be learned and generalized. More precisely, RC-TL clusters similar trajectory users and trains a single RL-CNN per cluster based on a few users' data, significantly

[1]Partially reproduced in this chapter – Copyright ©2011 IEEE.

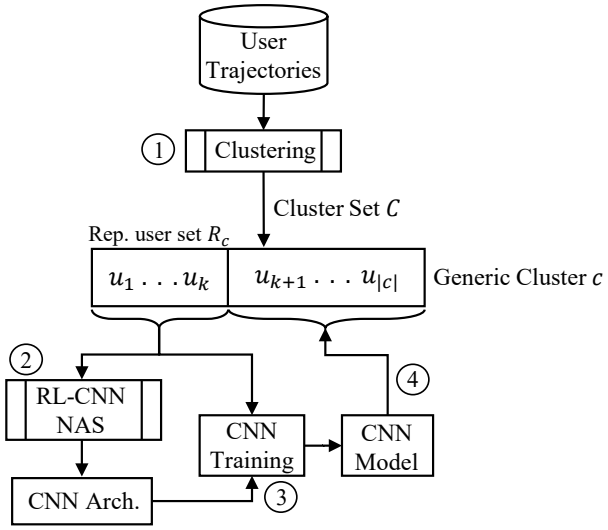



FIGURE 6.1: RC-TL trajectory predictor system architecture [21] ©2022 IEEE.

saving computation resources necessary to find the optimal architecture for such users. RC-TL provides an inter-cluster personalized yet intra-cluster generalized model.

To be more clear, RC-TL has three key features. First, it groups users with similar trajectories. Second, it applies a single RL agent to design a high-performance CNN architecture per cluster based on a few representative users, reducing the computation needed to train the model. Finally, it transfers the model trained on the cluster's set of representative users to the other members of that cluster using transfer learning, which is a technique for reusing one task's developed model as the starting point for another task to avoid initializing the second task from scratch and saving resources. By utilizing these techniques, RC-TL enables efficient and personalized individual trajectory prediction with reduced computational requirements in large-scale networks. The goal of RC-TL, is to predict in an efficient way the future trajectory for each user $u$ in the scenario, based on the user's past mobility data.

## 6.2 RC-TL Trajectory Prediction System Architecture

The RC-TL system operates in a set of sequential steps, briefly described hereafter and detailed in the following subsections. Figure 6.1 represents the system architecture and data flow. The circled numbers indicate the (1) clustering, (2) RL-CNN neural architecture search and training, and (3) TL steps of the system. The system's input is the database containing the user trajectories, and the system's output is one trained CNN model for each user in the scenario. The RC-TL's goal is to reduce the computational resources required to provide a CNN model for each user while keeping high prediction accuracy.

Algorithm 4 shows the overall RC-TL operation. The first step of the RC-TL architecture is the clustering, which retrieves user mobility information from a logically centralized database

**Algorithm 4:** RC-TL Trajectory Predictor [21] ©2022 IEEE

**Input:** Set of clusters $C$
**Output:** Trained CNN Models

**for** *each cluster* $c \in C$ **do**
  **for** *each user* $j \in c$ **do**
    Compute regularity ratio $\rho_j \leftarrow |T_j| / D_j$;
  $R_c \leftarrow \{k$ users with highest $\rho_j\}$;
  RL agent searches CNN architecture using $R_c$;
  Train CNN using $R_c$;
  Return prediction accuracy;
  **for** *each user* $j \in c$ **do**
    **if** *user* $j \notin R_c$ **then**
      Load pre-trained RL-CNN;
      user $j \leftarrow$ RL-CNN;
      Return prediction accuracy;
  Compute cluster's average prediction accuracy;
Compute overall average prediction accuracy over all clusters;

and groups users with similar trajectories in a set of disjoint clusters according to the LCSS similarity measure. For each cluster, this step also selects a subset of *representative users* who are associated with high-quality data by evaluating their data's coherence. The second step of the RC-TL architecture is the RL-CNN network architecture search, which can be run in parallel for each cluster. This step takes in input the trajectories of the cluster's representative users and uses an RL agent to find a set of hyper-parameters defining a CNN architecture that maximizes the trajectory-prediction accuracy for the representative users. After the agent selects the best CNN architecture, RC-TL trains it using the representative users' data and obtains the model's parameters in output. The fourth and last step of the RC-TL architecture is transfer learning, which takes into input the trained model's parameters and uses them to build a model for each of the other non-representative users of the cluster.

The performance of these steps is described in more details in next subsections.

### 6.2.1 User Trajectory Clustering and Reference Users Selection

In real-world scenarios, the number of mobile network users can be so large that training an individual RL-CNN trajectory predictor for each user is unfeasible due to the infrastructure's limited computing and storage resources. For this reason, we designed RC-TL to cluster users with similar trajectories using the LCSS algorithm [35]. Contrary to traditional Euclidean distance, the LCSS-based distance can be computed between trajectories made of a different number of data points $(t_i, b_i)$. RC-TL computes the LCSS distance $\gamma_{i,j} \in [0, 1]$ between every pair of trajectories $(T_i, T_j) \in \Theta^2$ in the dataset, and populates a symmetric distance matrix $\Gamma = (1 - \gamma_{i,j}) \in [0, 1]^{n \times n}$

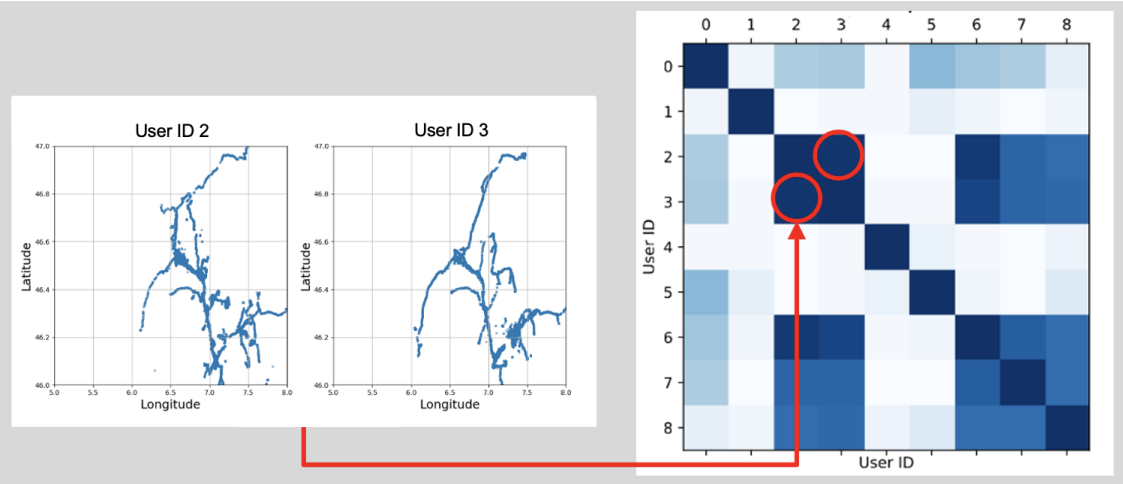



FIGURE 6.2: Two similar trajectories recognized by LCSS similarity Matrix.

that represents how different the trajectories are to one another. At this point, the system applies an unsupervised clustering algorithm to group the trajectories in a set $C = \{c_1, \ldots, c_q\}$ of disjoint clusters $c_i \in \Theta, \forall i \in \{1, \ldots, q\}$, where $c_i \cap c_j = \emptyset, \forall i, j \in \{1, \ldots, q\}, i \neq j$.

Clustering algorithms are designed to determine the optimal number of clusters and their members to minimize the average intra-cluster distance and maximize the inter-cluster distance. In other words, a suitable clustering algorithm must be designed so that all trajectories in a cluster have a small distance, whereas the trajectories belonging to different clusters have considerable distance. Therefore, any clustering algorithm whose geometry is based on distances between points, to compare pairwise the distances between locations within trajectories, can be a proper candidate, e.g., Birch, DBSCAN, K-Means, Mean-Shift, Ward, and Optics. Our proposed RC-TL system uses the Birch clustering algorithm, but any other clustering algorithm, such as K-means and Ward, can be used at this step [97]. The three clustering algorithms (K-Means, Ward, and Birch) are suggested as the design choices due to their out-performance concerning the other clustering methods in terms of accuracy and resource utilization.

The symmetric similarity matrix (correlation matrix) produced by the LCSS algorithm is depicted in Figure 6.2, along with two similar trajectories from the MDC[2] dataset. The level of similarity between pairs of user trajectories is indicated by the intensity of the blue color in the correlation matrix, with darker shades indicating higher similarity values. The correlation matrix exhibits a notable pattern, revealing a dense blue coloration between two real-life MDC trajectories. This observation indicates their strong similarity.

For each user cluster, the system selects a set of *reference users* that produced highly-descriptive mobility data, evaluated through our proposed regularity ratio metric, as presented in Section 4.6.

Users with a higher regularity ratio have visited a limited set of base stations multiple times, meaning that it is easier for a NN to infer a periodic behavior from such users compared to users with a low regularity ratio. The set $R_c$ of reference users for a generic cluster $c$ is selected as the

[2]https://www.idiap.ch/en/dataset/mdc

set of the $k = |R_c| << |c|$ users in the cluster with the highest regularity ratios $k$ is determined by a grid search that optimizes prediction accuracy.

### 6.2.2 Reinforcement Learning for CNN Architecture Design

As mentioned earlier in Section 2.4.2 and depicted in Figure 2.2, the architecture of 1D-CNN comprises multiple layers with various hyperparameters, such as kernel sizes, filter numbers, and pooling operations. These layers typically include convolutional layers, maxpool layers, flatten layers, dense layers, and dropout layers. These hyperparameters play a crucial role in the model's performance and its ability to capture spatio-temporal dependencies in mobility data.

To optimize 1D-CNN hyperparameters and improve prediction accuracy while reducing training time, we employ a reinforcement learning approach to search for high-performance networks tailored to the mobility characteristics of user data.

As the performance of RL is stated in Section 2.6, we define a search space for the RL agent to explore. The search space consists of multiple CNNs with different hyperparameter characteristics. The RL agent selects the optimal architecture from a finite and fixed *state space* of admissible architectures. The state space of RL-CNN, as discussed in Chapter 5, has a comparable size to that of RL-LSTM. This equivalence is crucial for ensuring a fair evaluation of the performance of both RL-CNN and RL-LSTM predictors. The objective of our study in this chapter is to showcase the superior performance and efficiency of CNNs for time series prediction over LSTMs. Therefore, it is imperative to establish a comparable search space between the two models. In RL-CNN, the state space is a subset of all the possible combinations of the values that the CNN hyper-parameters can assume. Namely, the state-space dimensions are:

- The number of layers that make up the CNN
- The type of each layer among convolutional, max-pooling, flatten, dense, and dropout
- The number and size of different kernels applied to each convolutional layer
- The stride of each max-pooling layer
- The number of perceptrons in each dense layer
- The dropout ratio of each dropout layer.

The RL-CNN state-space can become considerably large depending on the values of the mentioned parameters, making it impossible to test the performance of every CNN architecture in the space. As stated RL provides a method to search for the optimal architecture avoiding an exhaustive grid search.

To develop RL-designed, we assume that the CNN can contain at most one flatten layer, which divides the CNN in two subsequent sections with different purposes: *feature extraction* and *classification*. The feature extraction section can contain only convolutional and max-pooling layers, whereas the classification section can only contain dense and dropout layers. When optimizing the architecture of a CNN, it is important to consider the hyperparameters for both the feature

extraction and classification sections. These hyperparameters include the arrangement of layers, the number of layers, and the values of each layer's components.

As shown in Figure 2.4, at the beginning of an episode, at the beginning of an episode, the agent can take action $a$ from a subset $A(s)$ of the action space, where $A(s)$ depends on the currently observed state $s$. Every action always adds one layer to the current CNN and fixes the added layer's type and parameter values so that the action leads the environment (i.e., the CNN) into a new admissible state (i.e., architecture). In order to guarantee the arrival state is admissible, the new layer's type is constrained by a set of rules:

- First and last CNN 's layers must be a convolutional and a dense layer, respectively
- Convolutional and max-pooling layers can be followed only by another convolutional and max-pooling layer, or by a flatten layer
- A flatten layer can be followed only by a dense layer
- Dense layers can be followed only by other dense and dropout layers
- Dropout layers can be followed only by dense layers.

After the agent has taken action, the corresponding reward is unknown and must be computed by training the resulting CNN with the cluster's representative users' data for a limited amount of epochs (*exploration training*). The model's prediction accuracy on the mobility dataset is the reward associated with taking that action in that state. Similar to Section 5.2, the RL agent uses the *Q-learning* algorithm with an *ε-greedy* strategy to learn the policy for selecting actions and stops searching the state space for better CNN architectures when it has reached either a *target accuracy* or a maximum number of episodes. The reward for taking each action in each state is computed using the Bellman equation, presented in Equation 2.18.

During the initial phase of learning, the RL agent employs a strategy of random exploration within the search space. This approach allows the agent to discover potentially promising architectures. As the learning progresses, the agent gradually adjusts its policy to prioritize actions that lead to higher rewards, aiming to identify architectures that yield superior performance. At the conclusion of all episodes, the RL agent chooses the CNN architecture with the highest accuracy from among the architectures explored throughout the learning process. This selection is based on the agent's accumulated knowledge and experience, ultimately leading to the identification of the highest-performing architecture.

### 6.2.3 Transfer Learning between Cluster Members

After building and training the RL-CNN trajectory predictor associated with the cluster $c$, the RC-TL trajectory prediction could be constructed by transferring the pre-trained reference model from the $k$ reference users to the remaining $|c| - k$ users in the cluster $c$. Let the layers of the pre-trained reference model for cluster $c$ be $L = \{l_1, l_2, \ldots, l_h\}$, where the layers $l_1$ and $l_h$ represent input and output layers, respectively. The system transfers the learned knowledge of the reference model's neural architecture and weights of the first $h-1$ layers $\omega(l_j), \forall j \in \{1, \ldots, h-1\}$, to the remaining

TABLE 6.1: Fixed parameters of RC-TL System [21] ©2022 IEEE.

| Parameter | Values |
|---|---|
| Representative users per cluster | 10% |
| Batch size | 200 |
| Learning rate decay | 0.002 |
| Maximum training duration | 200 epochs |
| Early stopping patience | 10 epochs |
| Early stopping accuracy improvement threshold | 0.1 |
| Activation function in dense hidden layer | ReLU |
| Activation function in dense output layer | SoftMax |
| Discount factor $\gamma$ | 1 |
| Learning rate $\alpha$ | 0.01 |
| Early stopping threshold | 80% |
| Exploration training duration | 20 epochs |
| Exploration training validation | 10-fold cross-validation<br>70% data, 30% test |
| Exploration training target accuracy | 80% |
| Exploration training max episodes number | 500 |
| $\varepsilon_0$ | 0.01 |

$|c| - k$ users in the cluster $c$. RC-TL transfers the knowledge of the first $h - 1$ layers because the CNN's output layer requires a different number of classes (neurons) for each user. The $h$-th layer's number of neurons for the $j$-th user is determined from its mobility data, i.e., set equal to the $j$-th user's number $D_j$ of different base stations appearing in its trajectory. In this way, the remaining $|c| - k$ users initialize their NN with the transferred reference model and do not require to be trained from scratch.

## 6.3 Evaluations

### 6.3.1 Experimental Details

We test RC-TL's performance on a large-scale real-world mobility dataset provided by Orange S.A., France. As discussed in Section 4.4.1, the exact locations and identities of both the base stations and users in the Orange dataset are anonymized for privacy reasons. Therefore, we formulate the problem as a classification task where the objective of the predictor is to forecast the future base station IDs.

In our experimental setup, we fix some learning parameters for CNN and RL (see Table 6.1) and we define sets of CNN hyper-parameters as the RL search space (see Table 6.2). Each row corresponds to one of its dimensions. We selected such parameters and search space because of their popularity in the literature [6].

TABLE 6.2: CNN hyper-parameter search space in RC-TL System [21] ©2022 IEEE.

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

From Table 6.2, the number of hidden layers is chosen as an integer value within the range of $(4, 20)$, while the number of neurons in each hidden layer is selected from the discrete set $\{20, 40, 60, 80, 100, 150\}$. For the convolutional layers, we consider candidate values of the number of kernels from the set $\{46, 64, 128\}$ and the kernel size from the set $10, 20, 30$. To prevent overfitting, we explore a range of candidate values for the dropout ratio in the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. Furthermore, to enhance the non-linearity of each dense layer, we adopt the ReLU activation function. This wide range of hyperparameter choices ensures that we thoroughly search the architecture space to find the optimal RL-CNN model for individual user TP.

### 6.3.2 Evaluation Results

In this section, we perform two experiments. First, we evaluate the performance of the proposed RL-CNN against other state-of-the-art ML approaches without clustering, meaning that each neural network is built and trained for a single individual user. In this way, we focus on studying our proposed RL-CNN regarding the tradeoff between prediction accuracy and network build time. The competing predictors we implemented are:

- GS-LSTM and RL-LSTM, which are LSTM-based trajectory predictors whose architecture is determined through GS and RL, respectively.
- J48 and RF, which are non-neural decision-tree-based models.

Second, we evaluate RC-TL's performance employing three different clustering algorithms (k-means, Ward, and Birch [97]) against the non-clustered RL-CNN approach. In this way, we can evaluate the impact of clustering on prediction accuracy and the computational load required for training and study their tradeoff. Moreover, we can choose the best performing clustering algorithm. Each predictor's performance is evaluated and averaged on 100 random users.

The first experimental results show the impact of the neural network and neural architecture search on prediction accuracy and build time. We firstly compare all models on an individual, i.e., user-wise mobility prediction in which each algorithm predicts the user trajectory, and the accuracy is given in terms of the fraction of correctly predicted data points.

Figure 6.3 shows that RL-CNN achieves similar accuracy to RL-LSTM and around 10% higher than RF and J48. Meanwhile, RL-CNN saves 69% of the build time compared to RL-LSTM and is comparable with the RF's build time.

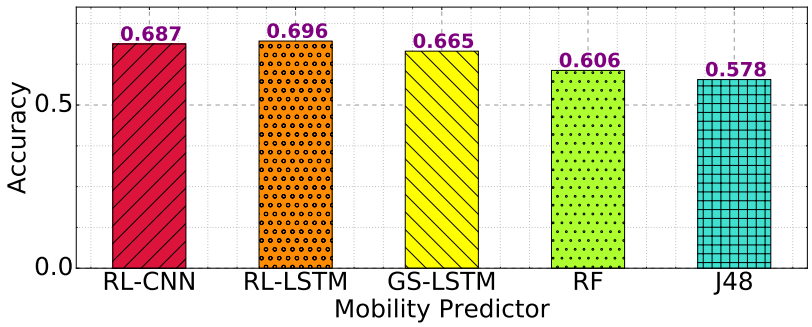



FIGURE 6.3: Accuracy of mobility predictors trained on an individual isolated user data [21] ©2022 IEEE.

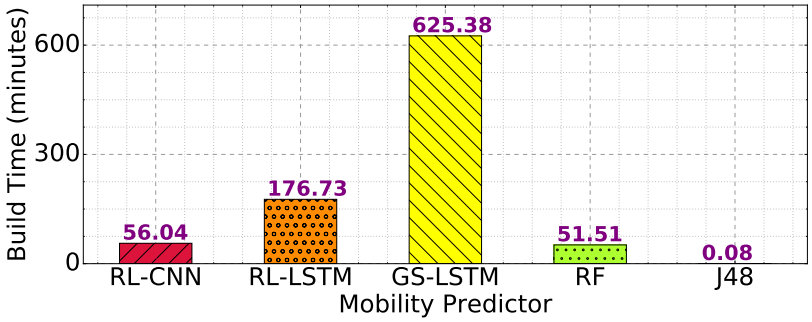



FIGURE 6.4: Build time of mobility predictors trained on an individual isolated user data [21] ©2022 IEEE.

Figure 6.4 also shows that RL reduces the build time (defined in Section 4.7.3) by 72% compared to grid-search neural network architecture search, with minimal difference inaccuracy. This behavior can be explained by the relatively short duration of the data collection for the dataset, spanning two months during summer holidays, incurring a more significant degree of exploration in the dataset, as users visit several new places to predict such features challenging. We expect our solution and its accuracy to improve for datasets with a more comprehensive data collection, as it will learn the statistical features of such data points.

We sample the learning curve over the episodes of transfer learning both in the case of applying CNN and LSTM networks in order to compare the convergence of both algorithms. Figure 6.5 shows that the RL agent can find CNN and LSTM architectures that achieve a 69% accuracy in both cases, even though the RL agent can converge to the best architecture for a CNN in fewer episodes than for an LSTM. This means less computation is necessary by the transfer learning step to find and evaluate NN architecture in order to converge to similar accuracy levels, showed in Figure 6.3.

We also sample the individual behavior of the learning process after an architecture has been defined. In Figure 6.6 we can see the accuracy of the NN by training epoch in both a CNN and

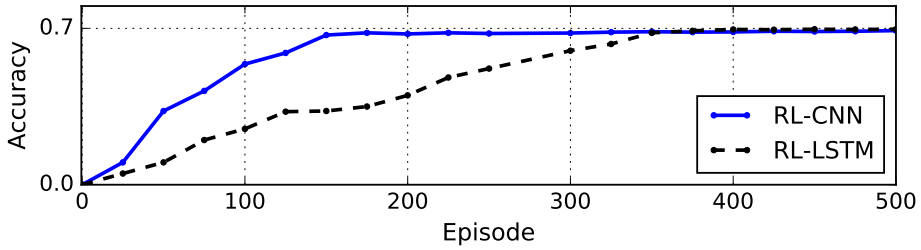



FIGURE 6.5: RL-designed ANNs curves for the average user in Orange dataset [21] ©2022 IEEE.

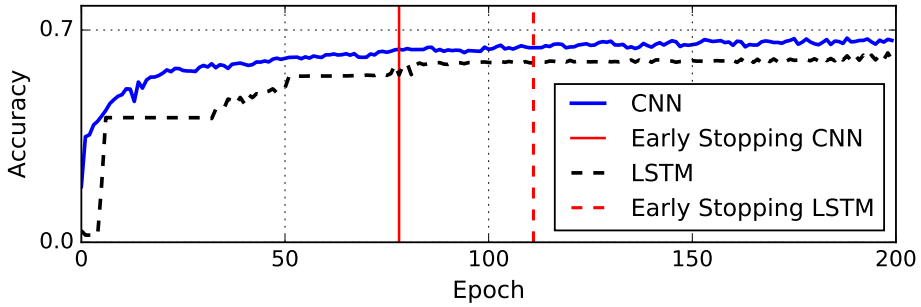



FIGURE 6.6: Learning curves for the best CNN and LSTM models selected by the RL agent for a random user [21] ©2022 IEEE.

an LSTM network. Moreover, Figure 6.6 shows that, between the best CNN and LSTM architectures selected by the RL agent, the CNN learns faster than the LSTM, meaning that it can reach higher accuracy on the test set in fewer epochs. This shows that in this particular case, the CNN architecture can learn the statistical features of user mobility faster and to a better degree than an equivalently chosen LSTM model. Thus, incurring a lower computational cost to train the architecture and, on a larger scale, better overall scalability of the network. We can conclude that CNNs whose architecture is searched by an RL agent achieve similar prediction accuracy to other state-of-the-art models and can be built in a fraction of the time required by other LSTM-based methods.

The results of the second experiment highlight the impact of clustering on the reduction of computational resource requirements for training. Figure 6.7 shows that the clustered RC-TL system achieves an almost identical prediction accuracy to the non-clustered RL-CNN predictor, with the Birch algorithm providing the best accuracy among the three tested clustering algorithms. RC-TL trains the model associated with the cluster using the data of 10% of users in the cluster with the highest regularity (representative users). This saves 90% of computational resources compared to training a dedicated model per user. It is worth noting that the Birch clustering algorithm detects half of the clusters detected by the k-means algorithm and 25% fewer clusters than the Ward algorithm, according to the *normalized number of clusters* metric (i.e., the ratio between the number of clusters detected by the considered algorithm and the highest number of clusters detected by

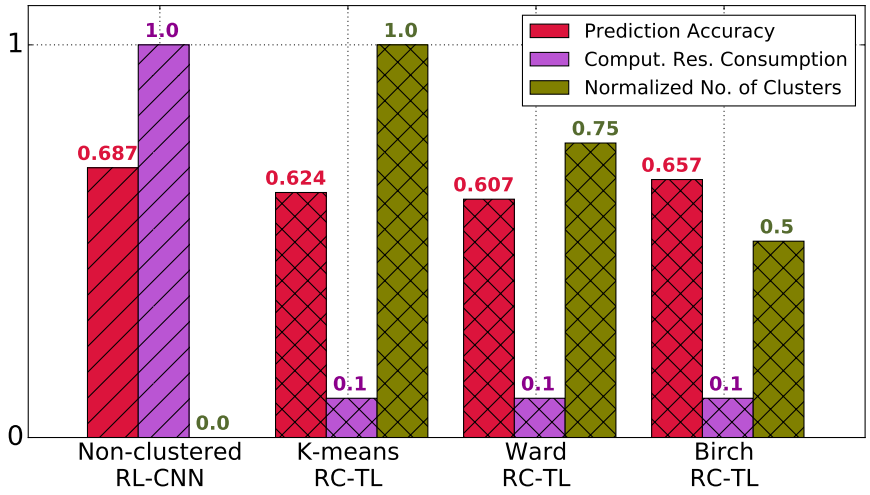



FIGURE 6.7: Performance of the non-clustered RL-CNN predictor, trained on a single user's data, compared with the clustered RC-TL [21] ©2022 IEEE.

TABLE 6.3: Impact of the number of representative users $k$ on accuracy and computational requirements of RC-TL [21] ©2022 IEEE

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

all algorithms). The number $k$ of reference users per cluster can be heuristically chosen by testing which values among 5%, 10%, and 20% of the cluster user leads to best accuracy and least computational requirements. Table 6.3 shows the impact of the possible values of $k$ on RC-TL's prediction accuracy and computational requirements. Training a single model per cluster with the data of 10% of users in the cluster and transferring the trained model's parameters to the other 90% of users in the cluster achieves much higher accuracy compared to training on 5% of cluster users and saves around a third of computational requirements compared to training the model on 20% of cluster users.

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

## 6.4 Chapter Conclusions

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

RC-TL demonstrates impressive capabilities in managing computational resource usage and handling the complexity associated with training personalized reinforcement learning-designed neural networks within large networks. However, a notable limitation of RC-TL is its exclusive consideration of an individual user's past location history, disregarding social interactions with other users. Although RL-CNN is trained separately for multiple clustered users of a large multi-agent network, it fails to account for the interdependencies and influences among different mobile users. While this approach can be effective in certain scenarios, it may not fully capture the intricate dynamics of social mobility patterns. In real networks, users influence each other's movements and adjust their behaviors based on the dynamics of a multi-agent environment. In the subsequent chapter (Chapter 7), we explore the incorporation of social factors into collaborative trajectory prediction models to enhance their accuracy and applicability in real-world scenarios.

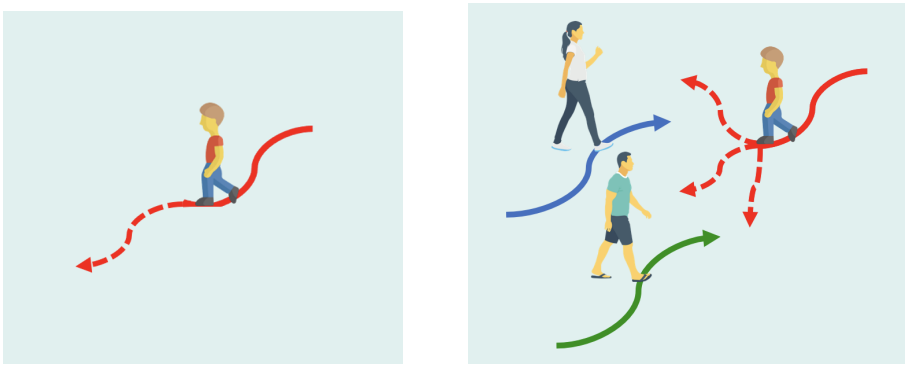
# Chapter 7

# Computationally-Efficient Multi-Agent Trajectory Prediction in Socio-Interactive Mobility Networks

## 7.1 Chapter Introduction

In the previous chapter, we proposed a reinforcement learning-based NN personalization solution for cluster-level trajectory prediction. We trained a model based on individual user characteristics, and leveraging the similarities between users within the same cluster, we transferred the pre-trained model to other cluster users. While our method is highly scalable, it lacks consideration of interdependencies among users within a cluster. Individual prediction and taking mobile users in isolation can be suboptimal for predicting complex motion behaviors of mobile users. This is because mobile users are often part of a larger social network and their behaviors are influenced by the spatio-temporal dependencies among neighboring mobile users' paths. In this direction, in this chapter, we extend the scope of cluster-level personalization to a social-aware collaborative trajectory prediction model that considers the interactions among users within a cluster.

Understanding social interactions and spatio-temporal dependencies can provide valuable information for predicting complex motion behaviors of mobile users. Social interactions can also affect the choices and behaviors of neighboring mobile users. For example, in *crowded scenarios*, this information can be leveraged for a wide range of applications, including collision avoidance, crowd management, activity recognition, and autonomous navigation. On the other hand, in *UE mobile networks* with geographically distant user trajectories and base station locations, social-aware trajectory prediction leverages social interactions and group movement patterns among users to enhance prediction accuracy and adaptability. By considering the collective intelligence, this approach optimizes resource allocation, enhance network performance, and improve user experiences [65]. Figure 7.1a depicts an individual user in an isolated-agent crowd scenario, where the user can take any path it wants and its future trajectory depends only on the history of its own visited location. However, this scenario is not very realistic since humans and agents naturally exist in multi-agent urban scenarios. On the other hand, Figure 7.1b shows interacting users in



(A) Isolated (B) Social

FIGURE 7.1: Comparison of mobile user trajectory prediction in single-agent isolation (a) and multi-agent collaboration (b) scenarios.

a multi-agent crowd scenario, where users must consider other users' decisions and states and respect each other's space to avoid collisions.

Despite the advances made in social-aware trajectory prediction using neural networks, these methods still face several other shortcomings. One concerning issue is that these models often either capture only local interactions within a fixed area around each agent, which can result in the neglect of larger, more distant interactions, or they consider all agents within a scene, which can lead to high computational costs. Thus, existing social-aware works are computationally inefficient. In this chapter, we aim to propose an efficient social interaction extraction techniques.

Another challenge, similar to Chapter 6, is that the design of the neural network architecture is typically based on human-based heuristics, which is a time-consuming and error-prone process. However, in the domain of social-aware mobility prediction, personalizing neural network architecture is more critical than personalizing individual users in isolated environments. This is because social interactions and behaviors are inherently collaborative and can involve complex dependencies among multiple users. Therefore, we aim to extend our RL-based NAS to the field of social-aware trajectory prediction to better capture the complex user interdependencies.

Moreover, in Chapter 6, we introduced 1D-CNNS as an alternative to RNNs for trajectory prediction, leveraging their ability to process data in parallel, making them more efficient. In multi-agent mobility scenarios, accurately modeling complex social interactions is even more crucial. Thus, in this chapter, we utilize Transformers neural networks as our trajectory predictor. Transformers are built on attention mechanisms that allow them to capture long-range dependencies of time-series data and model complex temporal relationships efficiently. The remarkable advantage of Transformers for time-series data prediction is their ability to handle variable-length sequences without the need for padding, enabling them to capture global context information from the entire sequence. This let Transformers make more informed predictions, particularly when long-term patterns significantly impact accurate forecasting.

In this chapter, we aim to address research questions stated in Section 1.2.3 as follows.

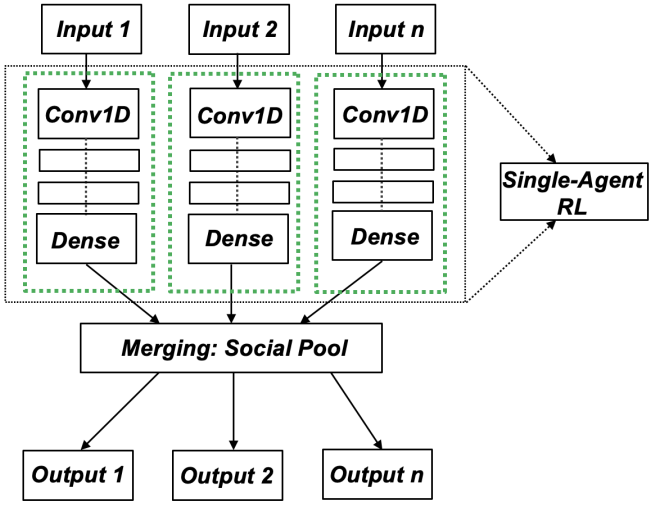



FIGURE 7.2: An overview of Reinforcement Learning-designed Social Learning for cooperative trajectory prediction.

*"RQ 3.1: How can we substitute widely adopted CNNs in trajectory prediction with alternative neural networks to improve the learning of intricate time-series mobility data more efficiently?".*

*"RQ 3.2: How can we decrease the computational complexity of existing social-aware methods while preserving the overall prediction accuracy of the system?".*

*"RQ 3.3: How can we incorporate the personalization paradigm of NAS into multi-agent social trajectory prediction, enabling it to search for high-performance neural networks specifically tailored to the distinct mobility patterns of interactive users rather than individual users?".*

Our objective is to overcome the aforementioned limitations by developing low-cost personalized trajectory predictors for multi-agent scenarios. Our goal is to accurately model the interactions among users, taking into account their specific similarities and mobile features while managing the computational budget. By doing so, we aim to capture the subtle influence they have on each other's mobility. In pursuit of these goals, we introduce a novel model: Intra-Cluster Collaborative Learning for Social Trajectory Prediction (INTRAFORCE) system [20][1].

In response to RQ 3.1, there is a need to explore alternative neural network architectures that can replace the commonly used RNNs or CNNs in social-aware multi-agent trajectory prediction. These alternatives should be capable of capturing complex spatiotemporal mobility features more efficiently. As a result, the first contribution of this chapter is the proposal of transformer neural network-based trajectory predictors. Recently, attention-based neural networks, particularly Transformers [86], have demonstrated significant improvements over RNNs. As described in detail in Section 2.4.3, transformers are designed to process sequences of data and utilize a self-attention mechanism to generate an output sequence by attending to different parts of the input

[1]Partially reproduced in this chapter – Copyright ©2011 IEEE.

sequence. This allows them to capture long-range dependencies more effectively than traditional recurrent neural networks.

Addressing RQ 3.2, it is imperative to address the computational complexity challenges of existing social-aware methods that inefficiently capture the impact of all users on each other. Our second contribution in this chapter focuses on mitigating this complexity. Specifically, we propose a clustering approach that groups together adjacent neighbor-trajectory users. By leveraging the mutual dependencies among intra-cluster users who are geographically clustered and exhibit significant influence on each other's movements and interactions, we aim to reduce the computational burden of current social-aware trajectory prediction methods. Our objective is to achieve this reduction in complexity while preserving the overall accuracy of the predictions.

In response to RQ 3.3, it is crucial to integrate the NAS personalization paradigm into social trajectory prediction. This integration enables the development of personalized multi-agent predictors that are specifically designed to accommodate the distinctive mobility patterns of interactive users. By considering the features and interactions among multiple users instead of solely focusing on an isolated user, the resulting predictors can better capture the complex dynamics of social mobility. This necessity arises from the fact that the utilization of transformers in mobility prediction is still relatively novel, and a notable limitation lies in the absence of efficient automation in designing transformer neural architectures within existing works. Consequently, the search space for well-performing transformer architectures remains relatively unexplored. In this regard, our third contribution of this chapter involves proposing an RL mechanism where the agent autonomously designs the neural architecture for a group of users with similar trajectories in a multiple-input multiple-output social model. In Figure 7.2, we compare our proposed approach to existing collaborative learning models depicted in Figure 2.6.

Considering the three aforementioned contributions, we introduce INTRAFORCE, an intelligent multi-agent trajectory prediction system. INTRAFORCE is a system to design and train *Social-Transformers* to capture joint interactions and reduce the required computation by measuring user trajectory similarity and clustering users before feeding them to a social-aware trajectory predictor to predict the joint user trajectories. For each cluster, INTRAFORCE employs a RL agent aiming at maximizing the performance of the multi-agent model based on the mobility features of the cluster. The goal of INTRAFORCE, is to predict the future trajectory for each user $u$, based on a user's past mobility data and intra-cluster users' mutual influences.

## 7.2 INTRAFORCE Trajectory Prediction System Architecture

INTRAFORCE operates through three modules: the *neighbor-trajectory user clustering*, the *reinforcement learning architecture search for transformer neural networks*, and the *intra-cluster social-transformer training* modules, represented in Figure 7.3 and explained hereafter.

Algorithm 5 provides a high-level description of the workflow of INTRAFORCE. The algorithm consists of three main sections that correspond to the three modules of the system. The first section of the algorithm (lines 1 to 4) describes the operation of the clustering module. This module clusters trajectory users based on their proximity in space using LCSS algorithm. The second section of the algorithm (lines 5 to 25) describes the operation of the architecture search module.

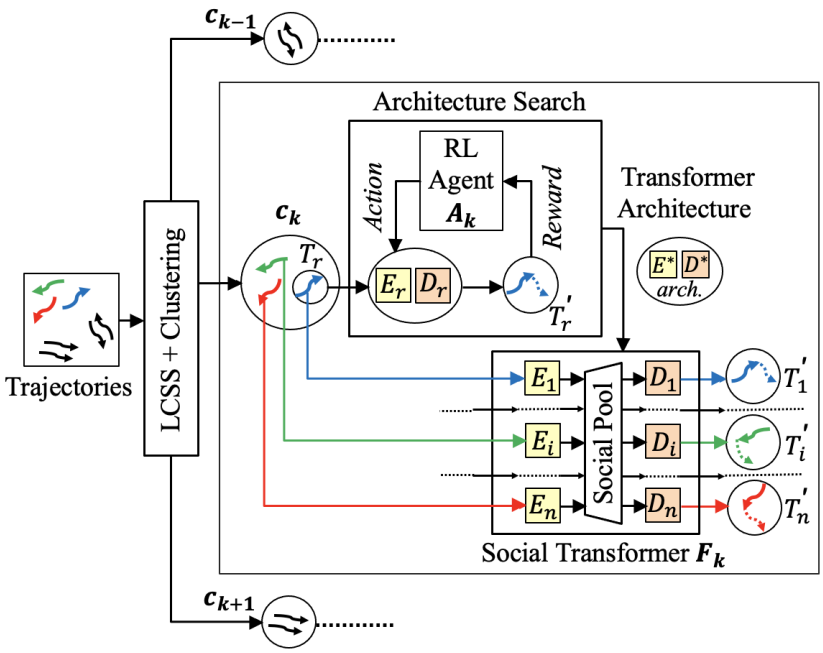



FIGURE 7.3: INTRAFORCE architecture [20] ©2022 IEEE.

This module uses a reinforcement learning algorithm to search for the optimal transformer neural network architecture for each cluster. The algorithm highlights all stages of training the RL as a NAS solution. The third section of the algorithm (lines 26 to 28) describes the operation of the social-transformer module. This module trains the selected neural network architectures using a social-pooling training algorithm. The social pool layer is able to capture the interactions among multiple mobile users moving through a dynamic environment.

### 7.2.1 Neighbor-Trajectory User Clustering

Similar to Section 6.2.1, this module groups users with similar trajectories by applying a clustering algorithm (e.g., Birch, DBSCAN, K-Means, and Ward) considering the LCSS distance [35] between their trajectories. The advantage of adopting LCSS is its ability to measure similarity between trajectories with different number of data points $p_u$.

### 7.2.2 Reinforcement Learning for Transformer Architecture Design

As mentioned earlier in Section 2.4.3 and shown in Figure 2.3, the transformer architecture consists of multiple Encoder Stacks $E$ and Decoder Stacks $D$. The Encoder Stack includes a positional encoding layer and several encoder layers responsible for converting a trajectory to an abstract representation. The Decoder Stack is responsible for generating a predicted trajectory from the learned abstract representations of user mobility and includes several decoding layers, dense layers, dropout layers, and an output layer (linear for regression and softmax for classification

**Algorithm 5:** INTRAFORCE Workflow [20] ©2022 IEEE

**Input:** Set of trajectories $\Theta$
**Output:** A Social-TF $F_k$ for each cluster $c_k \in C$

// Compute similarity matrix $A = (a_{ij})$ and set $C$ of clusters containing similar trajectories
**foreach** $(T_i, T_j) \in \Theta^2$ // Clustering Module
**do**
  $a_{ij} \leftarrow \texttt{LCSS}(T_i, T_j)$
$C \leftarrow \texttt{Clustering}(A)$ ;
// Build a social transformer $F_k$, $\forall c_k \in C$
**foreach** $c_k \in C$ // Architecture Search Module
**do**
  Elect the representative user $r_k$ for cluster $c_k$;
  // Initialize RL agent $A_k$ to optimize the Transformer architecture using data of user $r_k$
  $A_k \leftarrow \texttt{InitAgentRL}(\gamma, \alpha, \varepsilon, \varepsilon_0)$
  // Initialize state-action table to zero for all states and actions
  $\forall (s, a) \in S \times A : Q(s, a) \leftarrow 0$;
  // Initialize exploration probability $\varepsilon$ to maximum and empty architecture state
  $\varepsilon \leftarrow 1, s \leftarrow \emptyset$;
  // Optimize TF architecture up to $v_{\max}$ episodes
  **foreach** $v \in \{1, \ldots, v_{max}\}$ **do**
    // Decrease exploration every $v_{\max}\varepsilon_0$ episodes
    **if** $v \mod v_{max}\varepsilon_0 = 0$ **then**
      $\varepsilon \leftarrow \varepsilon - \varepsilon_0$;
    // $\varepsilon$-greedy strategy to select next TF architecture modification
    **if** $\texttt{RandomSample}([0, 1]) \leq \varepsilon$ **then**
      $a_v \leftarrow$ random action $a \in A(s)$;
    **else**
      $a_v \leftarrow \arg\max_{a \in A(s)} Q(s, a)$;
    // Update state according to action $a_v$
    $s' \leftarrow \texttt{UpdateState}(s, a_v)$;
    // Train the transformer with data of the representative user $r_k$ for a few epochs $\theta_s$. Compute model error $w$ and reward $\rho_v$ of architecture modification (action $a$)
    $s'_* \leftarrow \texttt{Train}(s', r_k, \theta_s)$;
    $w \leftarrow \texttt{ComputeModelError}(s'_*, r_k)$;
    $\rho_v \leftarrow 1/w$;
    // Update state-action table
    $Q(s, a_v) \leftarrow (1 - \alpha) Q(s, a_v) + \alpha \left( \rho_v + \gamma \max_{a \in A(s')} Q(s', a) \right)$;
    // Stop architecture search if model error $w$ (MSE for regression, Error Rate for classification) is below threshold $\eta$
    **if** $w \leq \eta$ **then**
      **exit loop**;
    $s \leftarrow s'$;
  // Selects the TF architecture $f_k$ with lowest model error
  $f_k \leftarrow \arg\max_{s \in S} Q(s, \cdot)$ // Social-TF Module
  // Initialize the Social-TF architecture for cluster $c_k$, with $n_k$ Encoder Stacks connected to $n_k$ Decoder Stacks through one social pool
  $F_k \leftarrow \texttt{InitSocialTF}(f_k)$;
  // Train the Social-TF $F_k$ with data from all $n_k$ users in $c_k$ for several epochs $\theta_1$
  $F_k^* \leftarrow \texttt{Train}(F_k, c_k, \theta_s)$;

problems). The encoder and decoder layers consist of multi-head attention modules, add and normalization layers, feedforward layers, dropout layers, and two residual connections. These hyperparameters are tunable and can be optimized during the neural architecture search process. The self-attention mechanism is a key component of the transformer architecture. The multi-head attention layer is a type of self-attention that allows the model to capture relationships between different segments of the input sequence. The multi-head module in the encoder and decoder layers enables parallel attention over distinct parts of the input sequence. The feedforward layer, on the other hand, learns the non-linear mapping between the input and output. Furthermore, transformers can be trained in parallel due to the self-attention mechanism, which leads to significantly reduced training times in comparison to recurrent neural networks. In the add and norm layer, the term *add* refers to the residual connection that prevents gradients from vanishing or exploding during deep neural network training.

This chapter uses $\varepsilon$-greedy $Q$-learning RL algorithm to select an optimal Transformer architecture for each cluster of users with adjacent trajectories who have maximum interactions with each other. By using this RL algorithm, the module is able to efficiently search for the best Transformer architecture that can accurately capture the unique mobility patterns and social interactions of each cluster of users. As stated in Chapter 6, in RL, an *agent* takes an *action* that has an impact on an *environment*, then observes the environment's *state*, and finally receives a *reward* from the environment. INTRAFORCE uses RL to select the optimal transformer architecture from a finite and fixed space of admissible architectures (the *state space*). In INTRAFORCE, the state space is a subset of all the possible combinations of the values of the social-aware transformer's hyperparameters. Namely, the state-space dimensions are:

- The number of Encoder and Decoder layers
- The number of multi-head attention layers
- The number and value of Add and Norm (normalization) layers
- The number of feed forward dense layers
- The number of perceptrons in each dense layer
- The dropout ratio of each dropout layer.

Each multi-head attention layer can have different numbers of heads $h$ and dimension of key. Feed-forward layers can have different numbers of neurons. Normalization layers can have different epsilon values for the encoder. Dropout layers can have different dropout ratios. The INTRAFORCE state space can become considerably large depending on the range of admissible values for the hyperparameters (e.g., see the RL action part of Table 7.1), so INTRAFORCE uses RL to search for the optimal architecture without performing an exhaustive grid search or a naive heuristic or random search.

After the agent has taken an action (i.e., selecting a candidate transformer model architecture), the reward associated to the selected architecture is unknown, and therefore, must be measured by training the transformer generated by the RL agent with the cluster's unique data for a few epochs

(*exploration phase*). At the end of the RL process, INTRAFORCE selects the transformer architecture with the highest accuracy among all those explored and completes its training until convergence (*exploitation phase*), over a larger number of epochs compared to the exploration epochs.

### 7.2.3 Intra-Cluster Social-Transformer Training

The INTRAFORCE module is responsible for predicting the trajectory of each user within a specific cluster by tuning an Encoder Stack per cluster user, pooling cluster users' mutual information in a Social Pool, and predicting one trajectory per cluster user using a Decoder Stack. This process involves training multiple encoders that use the same transformer architecture to convert trajectories into abstract representations, followed by pooling these representations to leverage the collective intelligence of the group. The Decoder Stack then generates predictions based on the intra-cluster user interactions, consisting of several decoding layers. The social pool method concatenates multiple neural networks to improve prediction accuracy by leveraging the collective knowledge of the different networks.

Overall, the training process in the INTRAFORCE system is optimized to efficiently capture multiple agents interactions while leveraging the power of transformer architecture and social pooling. Additionally, the social pool in INTRAFORCE only considers intra-cluster users, who share similar mobility patterns and have significant influence on each other's movements and interactions due to their geographic clustering. This approach reduces the computational costs of the system by focusing on a subset of eligible local users to participate in each federated training round, rather than processing data from all users in the scene regardless of their interaction patterns. By leveraging the mutual dependency of intra-cluster users, INTRAFORCE can accurately learn their movements and social interactions while maintaining overall system prediction accuracy.

## 7.3 Evaluations

We evaluate the performance of INTRAFORCE on the *small-scale* and a *large-scale* mobility scenarios, discussed in Section 4.4, against various state-of-the-art mobility predictors. The large-scale scenario is based on the *Orange* telecommunication S.A. [104] private dataset, while the small-scale scenario is based on the *ETH* [68] plus *UCY* [47] public camera dataset.

Mobile network datasets capture the trajectories of users across larger geographical areas, often spanning several kilometers. This characteristic of extensive coverage classifies datasets like the *orange* dataset as belonging to a *large-scale* scenario. In contrast, computer vision image-based crowd datasets focus on user trajectories within a smaller canvas, typically limited to a few meters. This limited spatial scope categorizes datasets such as the *ETH+UCY* dataset as representative of a *small-scale* scenario. By defining small-scale and large-scale scenarios, we aim to evaluate and demonstrate the superior performance of our trajectory predictors in different spatial contexts. This enables our predictors to be effective for both network management and crowd management goals.

Table 7.1 shows the parameters for the Transformer and RL agent training. The RL Agent Actions section of Table 7.1 shows the features of the search space for the hyperparameters that

TABLE 7.1: Experimental parameters for small-scale and large-scale scenarios in INTRAFORCE System [20] ©2022 IEEE

| Transformer Parameters | |
|---|---|
| Batch size (small-scale, large-scale) | 10, 200 |
| Learning rate decay | 0.002 |
| Social Transformer training epochs $\theta_l$ | 200 |
| Early stopping patience (in epochs) | 10 |
| Early stopping delta (small-scale, large-scale) | 0.05, 0.1 |
| Dense layers' activation func. (hidden, output) | ReLU, SoftMax |
| **Reinforcement Learning Parameters** | |
| Maximum RL training episodes $v_{\max}$ | 500 |
| Training epochs per episode $\theta_s$ | 20 |
| Discount factor $\gamma$, learning rate $\alpha$ | 1, 0.01 |
| Exploration rate decay $\varepsilon_0$ | 0.1 |
| Training target per episode (small-scale) $\eta$ | 0.05 |
| Training target per episode (large-scale) $\eta$ | 0.1 |
| Training validation (small-scale) | 4 sets train, 1 set test |
| Training validation (large-scale) | 10-fold, 70% train, 30% test |
| **RL Agent Actions: Transformer Hyperparameters Space** | |
| Number of hidden layers | $10, 11, \ldots, 50$ |
| Number $\xi$ of encoder and decoder layers | $1, 2, 3, 4, 5$ |
| Number of heads $h$ in a multi-head attention layer | $2, 4, 6, 8$ |
| Dimension of the key for a self-attention layer | $64, 128, 265$ |
| Normalization layer parameter | $10^{-2}, 10^{-3}, 10^{-6}$ |
| Number of perceptrons in dense layer | 20, 50, 80, 100, 150 |
| Dropout ratio in dropout layer | 0.15, 0.25, 0.5, 0.75 |

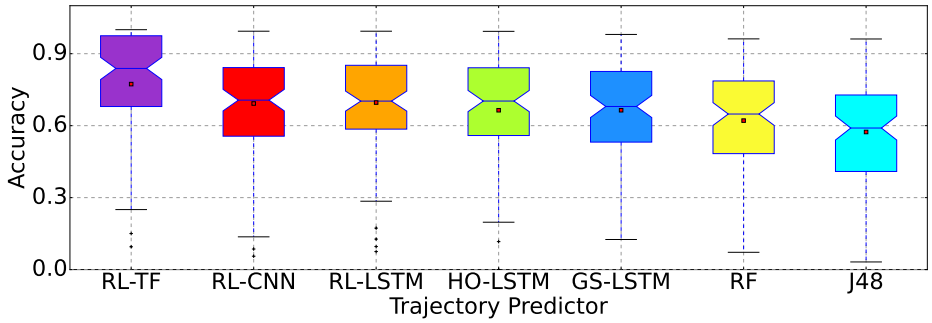



FIGURE 7.4: Accuracy different trajectory predictors [20] ©2022 IEEE.

defines the Transformer's architecture, where each row corresponds to one of the RL potential actions.

### 7.3.1 Large-Scale Mobility Scenario

We compare the performance of RL-TF with those of other state-of-the-art trajectory predictors, namely RL-CNN, RL-LSTM, HO-LSTM, GS-LSTM, RF, and J48. The first four baselines are neural-network-based predictors, while the last two, Random Forest (RF) and J48 Decision Tree, are non-neural predictors, which means that they do not require architecture search. RL-LSTM uses Reinforcement Learning to search for an optimal LSTM architecture, while HO-LSTM and GS-LSTM use Hyperopt (HO), an AutoML hyperparameter optimizer, and Grid Search (GS) for the same purpose. In the large-scale scenario, we perform two experiments. The first experiment compares the performance of a RL-TF with the aforementioned predictors in terms of prediction accuracy and build time for the individual user trajectory prediction, disregarding users' social interaction. The goal is to demonstrate the outperformance of Transformers concerning other machine learning predictors, and the outperformance of RL regarding other hyperparameter optimization models. The second experiment quantifies the impact of user clustering, cluster size, and the social-transformer model on accuracy, build time, and model size.

### 7.3.2 Small-Scale Mobility Scenario

We compare INTRAFORCE performance with those of popular social trajectory-predicion models, namely: Social-LSTM [3], Social-GAN [31], Sophie [74], Social-BiGAT [42], Social-Ways [4], Social-STGCNN [60], PECNet [56], and STAR [94]. In this scenario we perform one experiment in which we run the whole INTRAFORCE system, including user clustering, RL search for Transformer architecture, and the social-transformer trajectory prediction with social pooling, and then collect the Average Displacement Error (ADE) measurements.

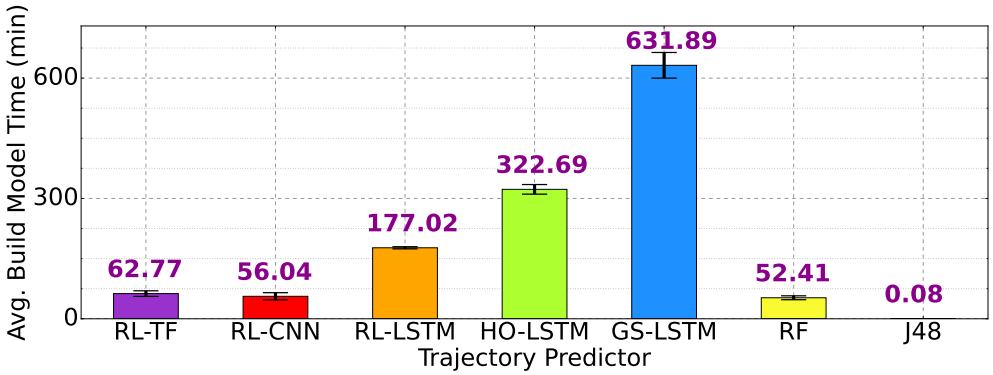



FIGURE 7.5: Build time of different trajectory predictors [20] ©2022 IEEE.

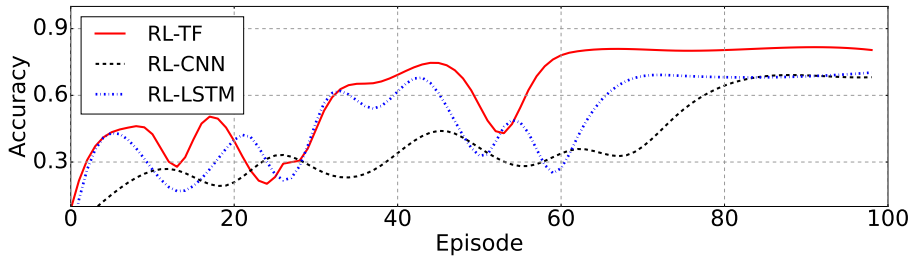



FIGURE 7.6: Accuracy convergence of the RL-designed predictors during the exploration and exploitation phases [20] ©2022 IEEE.

### 7.3.3 Large-Scale Evaluation Results

The results of the large-scale experiment (Orange dataset) show that using RL to search the Transformer's neural architecture leads to a higher prediction accuracy and a faster build model time compared to other trajectory prediction approaches. Furthermore, training one single social-transformer with a cluster's users data reduces the average training time and model size compared to training one separate model for each individual user.

We compare RL-TF with three NN predictors whose architectures are optimized by different NAS mechanisms and two none-neural predictors. Figure 7.4 shows that RL-TF achieves the highest mean accuracy (77%), which is 10% higher than the other reinforced models (RL-LSTM and RL-CNN) and almost 20% higher than non-neural models (RF and J48). We note that achieving higher prediction accuracy over the Orange dataset is limited by the restricted dataset size (63 days) and the huge diversity in users' data sample distributions. The accomplished accuracy of 77% is the average accuracy over 100 random users with acutely variable data quality and periodicity.

Figure 7.5 shows that RL-TF requires slightly above one hour of build time, which is similar to RL-CNN and RF. Although Transformers have larger architectures than CNNs and LSTMs, the Transformers' attention mechanism considerably reduces the build time. The RL-TF build time is similar to that of RF, which does not require architecture search but only training. RL-LSTM

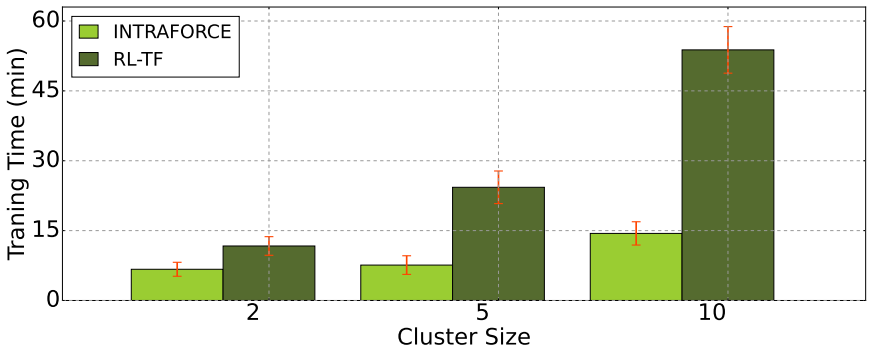



FIGURE 7.7: Average build time of individual RL-TF versus social INTRAFORCE trajectory prediction models [20] ©2022 IEEE.

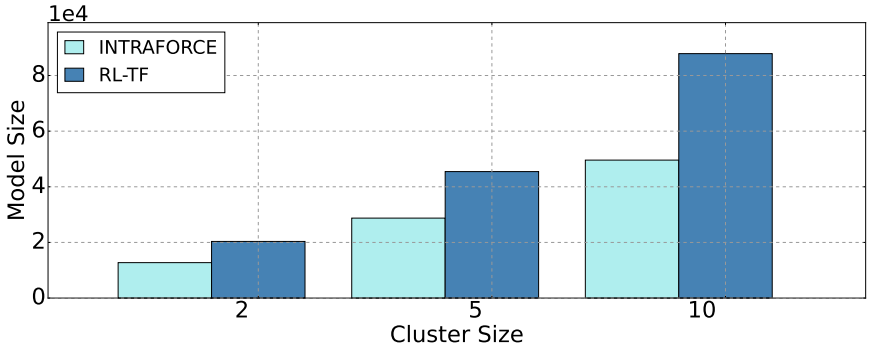



FIGURE 7.8: Average model size of individual RL-TF versus social INTRAFORCE trajectory prediction models [20] ©2022 IEEE.

and HO-LSTM require long build times due to the sequential nature of LSTMs, and the extensive training of HO exploration.

Figure 7.6 shows that the RL agent converges to a higher prediction accuracy in a shorter time with Transformer neural network, compared to LSTM and CNN.

Figure 7.7 and Figure 7.8 show the build time and the total model size needed by one social model for a cluster of users is lower than training an independent model for each user. As the number of users in a cluster increases, the difference in terms of build time and model size between training a separate model for each individual user (RL-TF) and training a single social-model per cluster of users (INTRAFORCE) dramatically increases. Moreover, we note that as the number of users in a cluster increases, social predictors achieve up to 5% higher accuracy, from 0.73 to 0.78, compared to individual predictors. We note that, The user clustering process within the social INTRAFORCE model is performed over the same set of 100 random users used in the individual RL-TF model.

### 7.3.4 Small-Scale Evaluation Results

According to the results of the small-scale experiment on the ETH and UCY datasets, using RL to search the neural architecture of Transformers leads to a lower training loss and subsequently a lower distance error between the predicted and actual trajectories. As shown in Table 7.2, INTRAFORCE outperformed several state-of-the-art social trajectory predictors, achieving the lowest average displacement error (ADE) of 0.22. These results demonstrate the effectiveness of INTRAFORCE's approach to trajectory prediction through the use of RL-based neural architecture search and the social pool method.

TABLE 7.2: ADE [m] of different social trajectory predictors for the small-scale scenario (ETH+UCY datasets) [20] ©2022 IEEE

| Work | ETH | Hotel | Univ | Zara1 | Zara2 | **Mean** |
|---|---|---|---|---|---|---|
| Social-LSTM | 1.09 | 0.79 | 0.67 | 0.47 | 0.56 | **0.72** |
| Social-GAN | 0.81 | 0.72 | 0.60 | 0.34 | 0.42 | **0.58** |
| SoPhie | 0.70 | 0.76 | 0.54 | 0.30 | 0.38 | **0.54** |
| Social-BiGAT | 0.69 | 0.49 | 0.55 | 0.30 | 0.36 | **0.48** |
| Social-Ways | 0.39 | 0.39 | 0.55 | 0.44 | 0.51 | **0.46** |
| Social-STGCNN | 0.64 | 0.49 | 0.44 | 0.34 | 0.30 | **0.44** |
| PECNet | 0.54 | 0.18 | 0.35 | 0.22 | 0.17 | **0.29** |
| STAR | 0.36 | 0.17 | 0.31 | 0.26 | 0.22 | **0.26** |
| **INTRAFORCE** | **0.31** | **0.24** | **0.22** | **0.14** | **0.23** | **0.22** |

## 7.4 Chapter Conclusions

We presented INTRAFORCE, a system to build a trajectory predictor that learns the social interaction within clusters of similar mobile users. INTRAFORCE uses Reinforcement Learning to build a Social-Transformer architecture based on the intra-cluster user mobility features. We evaluate INTRAFORCE on small and large scale scenarios, based on the ETH+UCY and the Orange datasets, respectively. In the small-scale scenario, INTRAFORCE achieves an ADE of 0.22, which corresponds to a lower positioning error compared to several state-of-the-art models. In the large-scale scenario, we show that Reinforced Transformers outperform LSTM- and CNN-based predictors by achieving up to $+10\%$ accuracy and up to $-70\%$ training time, and outperforms non-neural models based on RF and J48 of up to $+20\%$ accuracy. Our experiments show that increasing the number of users in a cluster leads to slightly higher accuracy, while considerably decreasing the time needed to build and train the trajectory predictors, as well as the number of training parameters.

Despite its significant advancements in accuracy improvement, reduction of mean squared error, and remarkable decrease in computational complexity, INTRAFORCE faces certain limitations. One such limitation is its centralized training approach, which involves collecting data and

may raise concerns regarding privacy issues and potential network bottlenecks when transmitting large amounts of data from users to servers. To address these limitations, the subsequent two chapters (Chapter 8 and Chapter 9) will focus on the development of multi-agent group intelligence using decentralized and distributed learning methods. These methods aim to enhance the scalability and privacy of the system by enabling agents to learn locally and communicate with each other in a distributed manner, thereby eliminating the need for a centralized server.

# Chapter 8

# Network-Adaptive Multi-Agent Trajectory Prediction in Distributed Mobility Networks

## 8.1 Chapter Introduction

In the previous chapter, we introduced a centralized social-aware trajectory predictor that utilizes personalized RL-designed Transformer neural architectures for each cluster, capturing the mobility features and intra-cluster interactions in trajectory prediction. Despite these advances, concerns about privacy and communication network limitations persist, particularly with regard to uploading large, private datasets to a centralized server and then performing collaborative learning. In this direction, in this chapter, we propose a distributed social-aware trajectory prediction.

The AI research community has introduced Federated Learning (FL) as a distributed ML framework [57]. FL provides a promising approach to overcome the limitations of centralized learning, allowing for more scalable and privacy-preserving multi-agent systems. In FL, the learning process is decentralized and takes place on the devices of individual users, which are connected to a central server. This approach allows users to keep their data private and avoid the need to transmit large datasets to a central location. Instead, the central server aggregates and processes the locally computed updates, while preserving the privacy of individual users' data. As described in Section 2.9, in an FL scenario, each client in the set of participants trains a local model using its local dataset and transmits the trained model's weights to a centralized server that aggregates them into a global model. After aggregation, the server redistributes the global model to the clients to retrain it and reduce the generalization error. A single back-and-forth training and weight exchange between FL clients and the central server is defined as a *training round*. The process of retraining and aggregation is repeated for $r$ consecutive training rounds until the global model converges.

FL-based trajectory prediction ensures that the confidential location data of mobile users remains on their devices, as the central server of the mobile operator only has access to base stations (macro mobility or large-scale dataset) and not the specific user geodetic coordinates (micro-mobility or small-scale dataset). Network providers seek to leverage prediction for different scales

of mobility. With access limited to base station information, the operators can only predict macro-mobility patterns. However, for accurate small-scale trajectory prediction, micro-mobility trajectories are necessary, which are sensitive and local users are hesitant to share with operators. In this context, FL offers a solution. Local users train models on their micro-mobility data and share only the trained model with the operator. This allows the operator to perform predictions without accessing the raw data, preserving user privacy while enabling effective mobility prediction for network optimization purposes [65].

Although the existing FL works have been successful in prediction tasks, they suffer from two main issues: (1) neural architecture inflexibility, and (2) ineligible client handling. In FL, neural architecture is inflexible because all clients share the same NN to execute a task, which might not fit the features of each user mobility dataset, computational resource availability, and the wireless network's communication conditions. For example, a model might be too large for some clients with reduced computation and communication resources, and too small for others with complex datasets.

Moreover, in FL, the aggregation server must collect the local models from all participating clients to compute a global model, which incurs computation delays due to local model training and communication delays due to model transmission. This method's limitation is that even if a single participating client is ineligible meaning that it has reduced computational or communication resources (defined as a *straggler*), the whole federated global model training process is slowed down. Likewise, the participation of ineligible low-quality data clients in federated training reduces the overall global accuracy and increases the whole system's computational resource consumption.

The objective of this chapter is to address the aforementioned limitations by seamlessly integrating the FL paradigm with RL-based personalization algorithms. This integration aims to achieve a high-performance decentralized collaborative system design while optimizing various objectives such as local users' computational budget, available throughput of wireless links, and the desired prediction accuracy set by the system through the RL-designed NAS. Furthermore, we introduce an intelligent model in this chapter specifically designed to filter eligible users as participants in federated training. The purpose of this model is to prevent the wastage of valuable bandwidth and resources by excluding users who may not contribute significantly to the training process.

In this chapter, we aim to address research questions stated in Section 1.2.4 as follows.

*"RQ 4.1: How can we develop a methodology to assess the quality of local user datasets, allowing this information to be conveyed to the central server in FL without giving access to the raw data?*".

*"RQ 4.2: How can we optimize computational resource utilization and reduce communication overhead in FL by strategically selecting a subset of eligible local users to participate in each federated training round?*".

*"RQ 4.3: How can we design an efficient multi-objective NAS algorithm for an FL setting that optimizes accuracy, training time, and transmission time, considering varying network throughput and federated participants' datasets?*".

In pursuit of these goals, we introduce a novel model: Network-adaptive Federated Learning for Reinforced Mobility Prediction (FedForce) system [18][1].

In response to RQ 4.1, facing the challenge in FL where the aggregating server does not have direct access to users' data, it is essential to invent a solution that can estimate the characteristics of local users' data and relay this information to the FL server without compromising data privacy. This requirement becomes crucial in developing an intelligent participant selection algorithm. By providing the server with an understanding of the data distribution across local users, it becomes possible to assess the eligibility of each participant and make informed selections accordingly.

As a result, the first contribution of this chapter is the proposal of a *regularity ration* metric in time and frequency domains. As described in detail in Section 4.6, we define the *time domain regularity ratio* for the $j$-th user $\rho_j^t$ as the ratio between the number of data samples in the data signal and the number of unique visited locations. While we define the *frequency domain regularity ratio* for the $j$-th user $\rho_j^f = S_j/F_j$ as the ratio between SNR of the data signal and dominant Frequency present. If both regularity ratios $\rho_j^t$ and $\rho_j^f$ meet a predefined threshold that we have set for data quality and periodicity, then the users are selected eligible. To determine the eligibility of users, both regularity ratios $\rho_j^t$ and $\rho_j^f$ are compared against a predefined threshold that we have established to assess data quality and periodicity. If both ratios exceed or meet this threshold, the users are considered eligible for participation.

In response to RQ 4.2, with the locally computed regularity ratio metrics from all local users that are transmitted to the server, we aim to minimize computational resource usage and communication overhead in FL. This will be achieved by selecting a subset of eligible local users for each federated training round based on a sorted list of eligibilities.

In response to RQ 4.3, there is a need for robust a NAS algorithm to personalize the neural network architecture in a manner that not only accommodates the unique characteristics of clients' mobility datasets, thereby increases the accuracy of the prediction task, but also considers the available computational and communication resources. Previously, in a centralized prediction setting, resources were abundant and there was no communication involved between the server and local users for the training phase. Thus, the objective of a NAS algorithm was solely focused on designing the highest-accuracy neural architecture tailored to each dataset. However, in distributed machine learning, where resources are constrained by IoT devices and wireless channels, factors such as resource availability and communication throughput become critical considerations. In this direction, the second contribution of this chapter focuses on proposing a multi-objective RL-based neural architecture search algorithm. This algorithm aims to optimize accuracy, training time, and transmission time for the proposed NN architecture, taking into account federated participants' datasets and varying network throughput. By achieving an optimal trade-off among these factors, the algorithm ensures the design of a federated NN architecture that fits the characteristics of clients' mobility datasets and utilizes available computational and communication resources effectively. This contribution is realized through FedForce, a RL-based system that automates the design process of the federated NN architecture.

[1]Partially reproduced in this chapter – Copyright ©2011 IEEE.

To be more clear, FedForce gathers regularity ratios and information regarding the computational resource availability of all system clients to let the server qualify users and chooses a set of eligible users to be engaged through the FL training phase. In FedForce, the RL agent is responsible for constructing a transformer neural network that optimizes the prediction accuracy while minimizing training time, model size, and transmission time. However, this optimization task is NP-hard since the agent must select the architecture from an enormous search space. This search space is even larger than that of RC-TL and INTRAFORCE trajectory predictors, which only focus on optimizing the NN's accuracy. In contrast, FedForce must optimize multiple objectives simultaneously, including accuracy, computational cost, and communication cost. RL makes the problem treatable by reducing the computations required to find the optimal solution compared to hyperparameter grid search.

Assuming that the mobility dataset contains a set of Independent and Identically Distributed (IID) users, FedForce distributes the RL-optimized NN among the FL participants to start training rounds. In the context of federated learning, IID means that each participant's data samples are independent and drawn from the same distribution. This assumption is important for ensuring that the model can generalize well across all participants' data. In FedForce, the set of user trajectories can be partitioned into a set of disjoint clusters $C = \{c_i \subseteq \Theta | i = 1, \ldots, q\}$, with $c_i \cap c_j = \emptyset, \forall i, j \in \{1, \ldots, q\}, i \neq j$, where each cluster contains IID trajectories. FedForce operates on each cluster independently, thus, an instance of the proposed method runs in parallel for each trajectory cluster. FedForce's goal is to predict the future trajectory for each user $u$, based on the user's past mobility data and other users' mobility information through the group intelligence provided in a distributed ML way.

## 8.2 FedForce Trajectory Prediction System Architecture

FedForce's workflow is divided into four phases, as shown in Figure 8.1 and detailed in Algorithm 6. These phases include: (1) local client eligibility estimation, (2) federated head client selection and RL-TF optimization, (3) federated participants selection and FL training, and (4) pre-trained model migration to silent clients.

The first section of the Algorithm 6 (lines 1 and 2) describes how eligible clients are estimated using our proposed regularity ratio metric. The second section (lines 3 to 19) describes how a head client is selected from a cluster of IID users and is responsible to train multiple RL agents for designing throughput-adaptive high-performance transformer neural architectures. Each RL agent optimizes the NAS problem from multiple perspectives of model accuracy, train time, transmission time over wireless channel, and model size. The third section of the algorithm (lines 20 to 23) describes how federated training is conducted among a set of eligible users. The server after receiving the optimized throughput-adaptive models from the RL agent, distributes them among all FL participants. Finally, the fourth section of the algorithm (lines 24 to 26) explains how the pre-trained network-adaptive models can be migrated to every user in the system including stragglers. This gives a decent opportunity of performing prediction tasks to every single user within the network.

**Algorithm 6:** FedForce Algorithm [18] ©2023 IEEE

**Input:** Set of trajectories $\Theta$
**Output:** A federated-TF NN $F_k$ for each FL participant $k$ and each silent user $u_j$

// 1. Client Eligibility Estimation Phase:
**foreach** $T_j \in \Theta$ **do**
  // Users compute regularity ratio locally and send to server
  $\rho_j \leftarrow |T_j| / D_j$;
// 2. Head Selection and RL-TF Optimization Phase:
// Build throughput-adaptive global TF $F_k$
$h_k \leftarrow \arg\max_{j \in \{1,\ldots,n\}} \rho_j$;
// Initialize RL agent $A_k$ to optimize the Transformer architecture using data of user $h_k$
$A_k \leftarrow \texttt{InitAgentRL}(\gamma, \alpha, \varepsilon, \varepsilon_0)$
// Initialize state-action table to zero for all states and actions
$\forall (s,a) \in S \times A : Q(s,a) \leftarrow 0$;
// Initialize exploration probability $\varepsilon$ to maximum and empty architecture state
$\varepsilon \leftarrow 1, s \leftarrow \emptyset$;
// Optimize one TF architecture per throughput class, up to $v_{\max}$ episodes
**foreach** $B \in [B_{low}, B_{mid}, B_{high}]$ **do**
  **foreach** $t \in \{1, \ldots, t_{max}\}$ **do**
    // Decrease exploration every $t_{\max}\varepsilon_0$ episodes
    **if** $t \mod t_{max}\varepsilon_0 = 0$ **then**
      $\varepsilon \leftarrow \varepsilon - \varepsilon_0$;
    // $\varepsilon$-greedy strategy to select next TF architecture modification
    $a_t \leftarrow \varepsilon\text{-}\texttt{GreedyAction}(\varepsilon)$;
    // Update state according to action $a_t$
    $s' \leftarrow \texttt{UpdateState}(s, a_t)$;
    // Train the transformer with data of the user $h_k$ for a few epochs $\theta_s$ and save model loss, training time, and size.
    $s'_* \leftarrow \texttt{Train}(s', h_k, \theta_s)$;
    $L_t, I_t, M_t \leftarrow \texttt{Evaluate}(s'_*, h_k)$;
    $X_t \leftarrow (L_t, M_t, M_t / B)$;
    $r_t \leftarrow -C_t = -\beta X_t$
    // Update state-action table
    $Q(s, a_t) \leftarrow (1-\alpha) Q(s, a_t) + \alpha \left( r_t + \gamma \max_{a \in A(s')} Q(s', a) \right)$;
    $s \leftarrow s'$;
  $H(B) \leftarrow s$
// 3. Participants Selection and FL Training Phase:
// Select $k$ users with largest $\rho_j$
$K \leftarrow \arg\max_{K' \subseteq \{\rho_1,\ldots,\rho_n\}, |K'|=k} \sum_{\rho \in K'} \rho$;
// Server distributes architectures $H(B_{\text{low}}), H(B_{\text{mid}}), H(B_{\text{high}})$ to users $\in K$
**foreach** $r \in R$ **do**
  $\texttt{FLStep}(K)$;
  // every $p$ FL rounds
  **if** $r \mod p = 0$ **then**
    // 4. Model Migration Phase: Server distributes the throughput-adaptive models $W(B_{\text{low}}), W(B_{\text{mid}}), W(B_{\text{high}})$ to all users
    **foreach** $j \in \{1, \ldots, n\}$ **do**
      // User $u_j$ determines its throughput interval and uses the associated model
      $B \leftarrow \texttt{MeasureThroughput}(u_j)$;
      $W_j \leftarrow W(B)$;

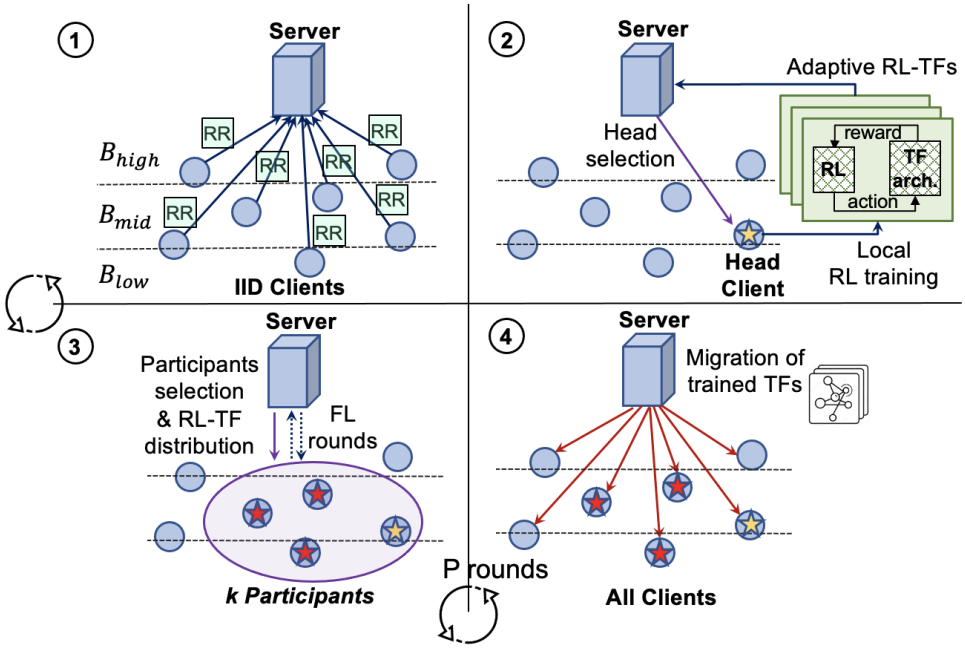



FIGURE 8.1: FedForce architecture and workflow [18] ©2023 IEEE.

### 8.2.1 Local Client Eligibility Estimation

During this phase, all users compute the regularity ratio metric, as proposed in Section 4.6, on their local data. The FL server collects users' regularity ratios and information regarding their computational resource capacity to estimate their data quality and periodicity letting only eligible clients participate in the training phase. This is an effective way to filter out the stragglers, who have inconsistent or poor-quality data or small computational resource available, from participating in the training phase. This approach helps save computational costs and training time, as only users with high-quality and consistent data are selected, ensuring better training outcomes. Moreover, since only the eligible users participate in the training phase, this reduces the communication overhead, thereby making the FL process more efficient.

### 8.2.2 Federated Head Client Selection and RL-TF Optimization

During this phase, the FL server chooses the user with the highest regularity ratio who has sufficient computational resources as the *head client*. The server asks the head client to train the RL agent and then, collects the optimized NN architecture and the trained weights.

At the head client side, the RL agent uses the $\varepsilon$-greedy $Q$-learning policy to design an optimal TF architecture that will be transferred to the server and distributed among FL participants.

In FedForce, the action space is the set of all possible combinations of the distributed multi-agent TF architecture's hyperparameter values. As stated in Chapter 7, TFs are composed of Encoder and Decoder stacks and characterized by a large set of hyperparameters named as the

number, characteristics, and sequence of multi-head attention layers, add and norm layers, feed-forward layers, and dropout layers. The FedForce action space can become remarkably large relative to the input range of the hyperparameters (see the RL Agent Actions section of Table 8.1).

At the beginning of each episode in FedForce, the RL agent proposes a TF architecture as its action, and the unknown reward associated with the selected architecture is evaluated by training the suggested TF on the selected *head client's data* for a few epochs. Once the *exploration phase* ends, the RL agent enters the *exploitation phase*, where it selects the TF architecture with the highest accuracy among all those explored. Similar to RC-TL and INTRAFORCE, the RL agent trains on the chosen architecture until the model converges over a larger number of epochs than the exploration phase. However, the RL algorithm in FedForce differs from the ones used in RC-TL and INTRAFORCE, as discussed in Chapter 6 and Chapter 7, as it has to optimize multiple objectives simultaneously in its cost function. We assume that on each RL episode $t$, the NN architecture proposed by the agent has a model size of $M_t$ [bit] $\in \mathbb{N}$, and achieves a loss $L_t \in \mathbb{R}$, which is defined as Sparse Classification Cross Entropy for classification and Mean Square Error for regression. We assume that during each episode $t$, each user holds an average wireless communication throughput $B_t$ [bit/s] with the centralized server. We define the time needed to train a NN at episode $t$ as $I_t$ [s] $\in \mathbb{R}$, and the time needed to transfer a NN model between client and server as $Y_t = M_t / B_t$ [s] $\in \mathbb{R}$. Finally, we define the *performance vector* $X_t = (L_t, I_t, Y_t)^\intercal \in \mathbb{R}^{3\times 1}$ as the vector whose components represent one system performance metric.

In FedForce, the RL agent's goal is to design a transformer NN architecture that jointly minimizes three objectives of model loss, training time, and model size (i.e., model transmission time), based on the information about the current communication throughput and the mobility dataset. We feed three throughput intervals to the RL agent to consider variable network conditions between clients and server. The RL agent is responsible to design three NN architectures per each of the throughput intervals so that if a mobile user changes its location during the FL training rounds, it still can access to a proper model.

Different applications have different requirements: (1) Autonomous driving applications require high prediction accuracy and ultra-low latency; (2) latency-sensitive network applications require faster prediction at the cost of a suboptimal accuracy; (3) traffic management services tolerate prediction delays but require the highest accuracy. Since it is not guaranteed that there is a single solution that minimizes all components of the performance vector simultaneously, the application must specify the optimizer parameters to achieve the desired tradeoff between model accuracy and training and transfer time.

We convert the aforementioned multi-objective optimization problem into a single-objective optimization problem by defining the RL's cost function $C_t = \beta X_t$ at episode $t$ as a linear combination of the prediction loss, training time, and transmission time regulated by the application coefficient $\beta = (\beta_1, \beta_2, \beta_3) \in [0,1]^{1\times 3}$ where, $\|\beta\|_1 = \sum_{i=1}^{3} |\beta_i| = \beta_1 + \beta_2 + \beta_3 = 1$. The larger the $i$-th coefficient component $\beta_i$ is in the cost function $C_t = \beta_1 L_t + \beta_2 I_t + \beta_3 Y_t$, the higher the optimization algorithm prioritizes the related variable. At each episode $t$, the RL agent uses a reward function $r_t = -C_t$, to evaluate the performance of the candidate NN architecture and updates its Q-table using the Bellman equation (Equation 8.1), where $\gamma \in [(]0, 1)$ is the *discount factor* that regulates the relevance of recent rewards, $s'$ is the resulting state from the action selected by the agent, and $\mathcal{R}(s')$ is the action space from state $s'$. We define the *cumulative reward* $r_t^*$ as

$r_t^* = \sum_{k=0}^{T'} \gamma^k r_{t+k}$, where $T'$ is the number of episodes used to compute the cumulative reward. As the number of episodes increases, the Q-table's content corresponding to the optimal policy $\pi$ tends to $Q^\pi(s_t, a_t) = \mathbb{E}[r_t^* | s_t, a_t]$. The RL agent selects the best NN architecture by searching the state $s^*$ corresponding to the maximum value of the Q-table (i.e., the expected cumulative reward), as shown in Equation 8.2.

$$Q_{t+1}(s,a) = (1-\alpha) Q_t(s,a) + \alpha \left( r_t + \gamma \max_{a' \in \mathcal{R}(s')} Q_t(s', a') \right) \tag{8.1}$$

$$s^* = \left( \underset{(s,a) \in \mathcal{S} \times \mathcal{R}(s)}{\arg\max} Q^\pi(s,a) \right)_1 \tag{8.2}$$

### 8.2.3 Efficient Participant Selection and Federated Training

During this phase, the FL central server receives the regularity ratio of each local user, which is a numerical value that can be transmitted efficiently without violating privacy. The server then sorts the ratios in descending order and selects a subset of $k$ users with the highest ratios, so that $k << |n|$, and sufficient computational resources to participate in the federated training rounds. The server distributes the RL-personalized throughput-adaptive NN architectures to these selected users to initialize their local models. Each local user then selects and adapts the best neural architecture based on their available uplink throughput and starts training. After training, users only share their trained weights with the server, rather than transmitting heavy and private data. At the end of each training round $r$, the server collects the trained weights from all clients and aggregates them into a global model per throughput interval. The server then returns the updated model to the selected $k$ local clients to continue local training until convergence is achieved.

Since mobile users are free to move within the defined urban scenario, the server distributes all three global models to every FL participant on each communication round $r$, so that they could choose the appropriate architecture based on their distance from the server and the signal strength they receive defining their throughput. We note that the value of $k$ is determined by a grid search that optimizes the tradeoff between prediction accuracy and computational resource consumption.

### 8.2.4 Pre-trained Model Migration to Silent Clients

During this phase, the server periodically, every $p$ rounds out of total training rounds, transfers all three trained throughput-adaptive global models to every $n - k$ system user, including ineligible users who were not participating in training who we call them silent users. FedForce system oscillates between phases 3 and 4 every $p$ rounds to let silent users catch up with the trained global models. We note that the value of $p$ is determined through a grid search that optimizes the tradeoff between prediction accuracy and communication overhead.

During this phase, the server transfers all three trained throughput-adaptive global models to every $n - k$ system user, including those who were not eligible to participate in training, known as silent users. This process occurs periodically every $p$ rounds, meaning that FedForce system

switches between phases 3 and 4 every $p$ rounds to ensure that silent users have access to the trained global models. This approach enables silent users to catch up with the latest model updates, which ultimately improves prediction accuracy and system performance. The value of $p$ is determined through a grid search that optimizes the balance between prediction accuracy and communication overhead.

## 8.3 Evaluations

We evaluate FedForce's performance on the *small-scale* and a *large-scale* mobility scenarios, as discussed in Section 4.4, against various state-of-the-art mobility predictors. The large-scale scenario is based on the *Orange* telecommunication S.A. [104] private dataset, while the small-scale scenario is based on the *ETH* [68] plus *UCY* [47] public camera dataset.

As discussed in Section 4.7.1, we model the average wireless channel throughput $B_t$ between clients and the server at each time slot $t$ as a uniform distribution from 10 Mbit/s to 100 Mbit/s, formally $B_t = \mathcal{U}([10, 100]), \forall t \in \mathbb{N}$. We categorize the throughput values into three intervals: low throughput, from 10 Mbit/s to 40 Mbit/s, medium throughput, from 40 Mbit/s to 70 Mbit/s, and high throughput, from 70 Mbit/s to 100 Mbit/s. While we have chosen three intervals for simplicity, FedForce can be extended to accommodate more throughput intervals if necessary.

The third section of Table 8.1 (RL Agent Actions) shows the search space of hyperparameters that form the TF architecture in the decentralized learning FedForce System, where each row corresponds to one of the RL's potential actions.

### 8.3.1 Large-Scale Mobility Scenario

In the *large-scale* scenario, to evaluate the FedForce performance in predicting the future base station IDs for each user (defined as a classification task), we quantify the impact of RR-based federated client selection and multi-objective optimized RL-TF, within the FedForce system, with respect to the classical FL algorithm and centralized ML trajectory predictor, on various metrics of accuracy, train and transmission time (model size), resource consumption, and communication overhead.

### 8.3.2 Small-Scale Mobility Scenario

In the *small-scale* scenario, to evaluate the FedForce performance in predicting users' future location coordinates (defined as a regression task), we compare it with group intelligence-based trajectory predictors (either social-based central predictors or federated-based distributed predictors), namely: Social-LSTM [3], Social-GAN [31], Sophie [74], Social-Ways [4], Social-STGCNN [60], STAR [94], INTRAFORCE [20], and ATPFL [87] in terms of average displacement error.

### 8.3.3 Large-Scale Evaluation Results

The results of the large-scale experiment (Orange dataset), Figures 8.2 to 8.7, show that using RL to search the TF's architecture leads to a higher prediction accuracy and a faster neural architecture

TABLE 8.1: Experimental parameters for small-scale and large-scale scenarios in Fed-Force System [18] ©2023 IEEE

| Fixed Federated Learning and Transformer Parameters | |
|---|---|
| FL training rounds | 100 |
| Clients' training epochs | 100 |
| Batch size (small-scale, large-scale) | 10, 200 |
| Learning rate decay | 0.002 |
| Early stopping patience (in epochs) | 10 |
| Early stopping threshold (small-scale, large-scale) | 0.05, 0.1 |
| Dense layers' activation func. (hidden, output) | ReLU, SoftMax |
| **Fixed RL Parameters** | |
| Maximum RL training episodes $v_{\max}$ | 500 |
| Training epochs per episode (exploration) $\theta_s$ | 20 |
| Training epochs after exploitation $\theta_l$ | 200 |
| Discount factor $\gamma$, learning rate $\alpha$ | 1, 0.01 |
| Exploration rate decay $\varepsilon_0$ | 0.1 |
| Training target per episode (small-scale) $\eta$ | 0.05 |
| Training target per episode (large-scale) $\eta$ | 0.1 |
| Cost function $C$ coefficient $\beta = (\beta_1, \beta_2, \beta_3)$ | $(1/3, 1/3, 1/3)$ |
| Exploration training validation (small-scale) | 4 sets train, 1 set test |
| Exploration training validation (large-scale) | 10-fold, 70% train, 30% test |
| **RL Agent Actions: Transformer Hyperparameters Space** | |
| Maximum mumber of hidden layers | $10, 15, \ldots, 50$ |
| Number of encoder and decoder layers | $1, 2, 3, 4, 5$ |
| Number of heads in a multi-head attention layer | $2, 4, 6, 8$ |
| Dimension of the key for a self-attention layer | $64, 128, 265$ |
| Normalization layer parameter | $10^{-2}, 10^{-3}, 10^{-6}$ |
| Number of perceptrons in dense layer | 20, 50, 80, 100, 150 |
| Dropout ratio in dropout layer | 0.15, 0.25, 0.5, 0.75 |
| **Mobile Networking Parameters** | |
| Number of users | 100 |

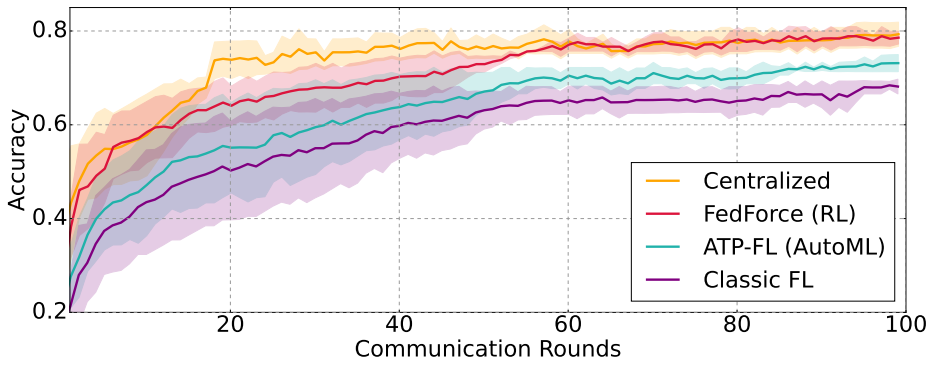



FIGURE 8.2: Accuracy convergence over federated training rounds of FedForce for the large-scale scenario (Orange dataset) [18] ©2023 IEEE.

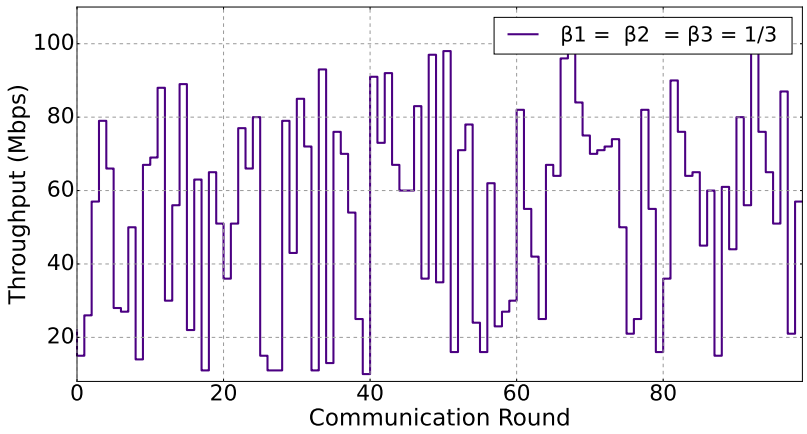



FIGURE 8.3: Time-varying wireless network bandwidth of FedForce for the large-scale scenario (Orange dataset).

search and train time compared to centralized TP approaches. Furthermore, FedForce achieves a better tradeoff over prediction accuracy, training and transmission time, resource consumption, and communication overhead compared to the classical FL used in other TP works.

In particular, Figure 8.2 presents a comparison of the convergence rates of accuracy in FedForce with respect to classical FL (without NAS), ATPFL (which uses AutoML NAS to design high-accuracy NNs), and centralized ML. The plot also includes 95% confidence intervals to provide a statistical measure of the variability in the results. We can observe that FedForce can converge to a similar average accuracy compared to the centralized model although experiencing a small delay in reaching the plateau. Moreover, we observe that FedForce achieves 10% higher prediction accuracy than the classical FL model and 7% higher than ATPFL.

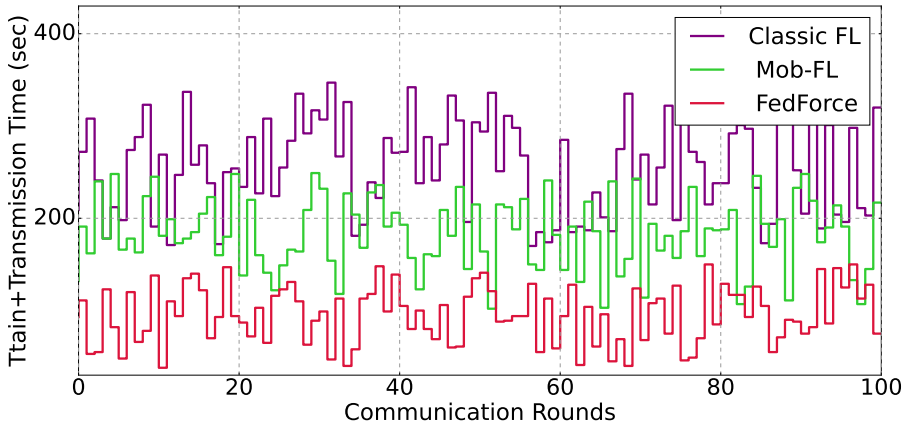



FIGURE 8.4: Train plus transmission time of FedForce over varying network throughput for the large-scale scenario (Orange dataset) [18] ©2023 IEEE.

Wireless channels are subject to a variety of environmental factors that can affect their performance and characteristics over time. These factors include interference from other devices, physical obstructions, distance between devices, and variations in temperature and humidity. As a result, the wireless channel's performance can vary significantly over time, leading to fluctuations in channel throughput. This is illustrated in Figure 8.3, which shows the wireless channel throughput modeled through a uniform random distribution over the FL communication rounds. The fluctuations in channel throughput depicted in the figure demonstrate the importance of adaptive transmission protocols and dynamic resource allocation schemes, such as those employed in FedForce, that can adapt to the varying characteristics of wireless channels to improve overall system performance.

For fair comparisons, Figure 8.4 compares FedForce's training plus transmission time to classical FL (without resource management) and MOB-FL (which manages computational resource usage). Figure 8.4 shows that the overall latency of training plus transmission time for FedForce is almost 50% less than MOB-FL and 70% less than the classical FL over a time-varying wireless network bandwidth presented in Figure 8.3. The training and transmission time for classical FL and Mob-FL is 265 seconds and 182 seconds, respectively, whereas for FedForce is only 97 seconds.

In Figure 8.5 and Figure 8.6, FF and FL stand for FedForce and Classical-FL, respectively. Figure 8.5 shows the impact of the number of federated participants (denoted as $k$) on the tradeoff between resource consumption and accuracy. FedForce achieves comparable performance to classical FL, which uses 100% of the system users ($k = 100$) for training, even with the smallest number of participants (i.e., $k = 10$), thanks to the use of RL in designing high-performance neural networks. In contrast, the global accuracy remarkably decreases in classical FL with $k = 30$, not only due to the absence of NAS but also because the $k = 30$ users are chosen randomly, while in FedForce, eligible users are selected based on estimated local data qualities through our proposed *regularity ratio* metric.

Figure 8.6 shows the impact of the number of global model migrations (denoted as $p$), from

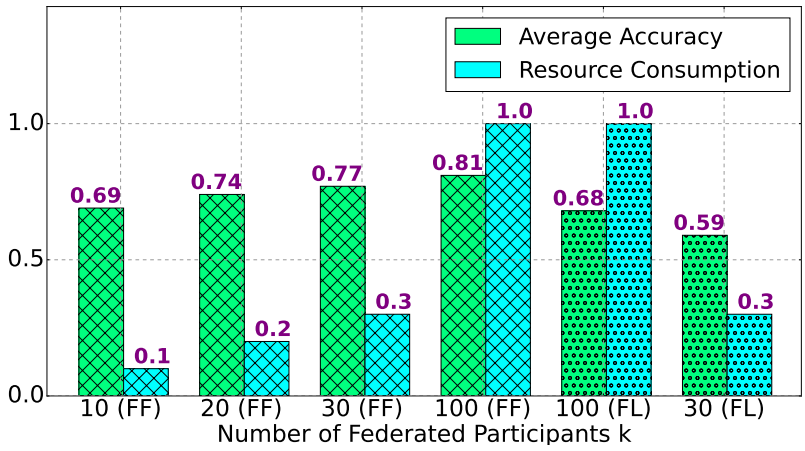



FIGURE 8.5: Choice of number of federated participants $k$ in the FedForce system for the large-scale scenario (Orange dataset) [18] ©2023 IEEE.

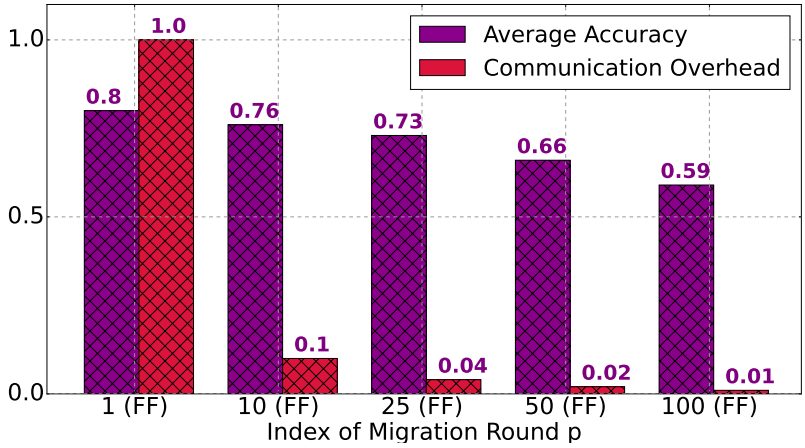



FIGURE 8.6: Choice of number of migration rounds $p$ in the FedForce system for the large-scale scenario (Orange dataset) [18] ©2023 IEEE.

$k$ federated participants to $n - k$, silent users, on the tradeoff between communication overhead and accuracy. We observe as the frequency of migrating the trained global models increases (from right to left), the system's communication overhead also increases. Consequently, since all users receive more frequent model updates, the overall system accuracy increases as well. We can see that FedForce achieves an average prediction accuracy of 76% by choosing $k = 30$ federated users, and migrating global models to silent users at every $p = 10$ rounds (meaning to have total of 10 migrations through 100 training rounds).

FedForce can save up to 80% of computational resources by choice of $k = 20$ and up to 96% of

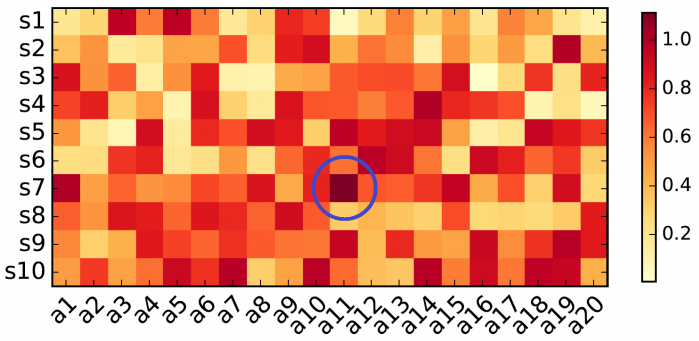



FIGURE 8.7: $Q$ table of RL, adopting Q-learning policy, with normalized $Q(s,a)$ values, corresponding to the FedForce cost function $C_t$ [18] ©2023 IEEE.

communication overheads by choice of $p = 25$ (total of 4 migrations) with an acceptable accuracy drop to 72% and 73%, respectively.

Figure 8.7 pictures the $Q(s,a)$ table computed through Q-learning policy over RL. Each small square shows a normalized Q value corresponding to the accumulated reward of a certain state-action. In this heat map, the color density corresponds to the Q-value or, more precisely, the cumulative reward associated with a specific action taken at a particular state. A higher density color indicates a higher Q-value, which indicates a more favorable or rewarding outcome for the corresponding action in the given state. For this particular RL example, we observe that the NN architecture corresponding to the $Q(s_7, a_{11})$ (RL agent's 11th action when RL system has been in 7th state) achieves the maximum reward or utility function. We note that, FedForce performance shown in Figures 8.2, 8.4, 8.5, and 8.6), are accomplished through the neural architecture choice of RL agent as stated in Figure 8.7.

### 8.3.4 Small-Scale Evaluation Results

Table 8.2 shows the results of the small-scale experiment (ETH+UCY datasets), in which FedForce achieves the lowest ADE (0.20m) compared to several state-of-the-art centralized social trajectory predictors and ATPFL [87] distributed social trajectory predictor. s ATPFL is the only work reported in the literature that employs the popular ETH+UCY social-interactive mobility datasets for the trajectory forecasting through federated learning. Aditionally, it can be observed that the weighted FedAvg algorithm performs better than the normal FedAvg algorithm in reducing displacement error.

## 8.4 Chapter Conclusions

In this chapter, we proposed FedForce, a federated mobility and trajectory predictor that uses RL to design a transformer architecture based on the unique mobility features, the computational resource availability, and the wireless network throughput. FedForce engages only a small group

TABLE 8.2: ADE [m] of different social and federated trajectory predictors for the small-scale scenario (ETH+UCY datasets) [18] ©2023 IEEE.

| Trajectory Predictor | NN | NAS | FL | **ADE** |
|---|---|---|---|---|
| Social-LSTM [3] | LSTM | - | - | **0.72** |
| Social-GAN [31] | GAN | - | - | **0.58** |
| SoPhie [74] | GAN | - | - | **0.54** |
| Social-Ways [4] | GAN | - | - | **0.46** |
| Social-STGCNN [60] | GCNN | - | - | **0.44** |
| ATPFL [87] | GNN-RNN | Auto-ML | FedAvg | **0.30** |
| STAR [94] | Transformer | - | - | **0.26** |
| FedForce | Transformer | RL | FedAvg | **0.26** |
| INTRAFORCE [20] | Transformer | RL | - | **0.22** |
| **FedForce** | **Transformer** | **RL** | **Weighted FedAvg** | **0.20** |

of users who posses high quality of data and training capacity to participate in federated rounds to guarantee efficiency in computational and communication costs.

We evaluated FedForce on two different datasets and showed that FedForce converges to the same accuracy as the centralized training model, while the classical FL is up to 10% less accurate and takes 50% more training and transmission time. Moreover, FedForce achieves an ADE of 0.20m, which is lower than the best-performing TP baseline. FedForce also can save up to 80% of computational resources and 96% of communication overhead without remarkably reducing the mean accuracy.

While FedForce has demonstrated robustness as a trajectory predictor, ensuring privacy, low communication overhead, manageable resource usage, and high prediction accuracy, it operates under the assumption that all local clients exhibit cooperative behaviors and employ a single neural network, designed by the head client, for all federated clients. However, in reality, even within a group of IID mobile users as considered in FedForce, there can be competition among users, distinct mobility features, and diverse desires. It is important to note that federated participants, despite having IID datasets, may possess different mobility characteristics. In the context of a distributed multi-agent scenario, it is probable that different users have distinct desires and, as a result, require their own personalized neural architectures to enhance their performance and efficiency. Therefore, in the next Chapter 9, we aim to model distributed mobile users in a non-cooperative simultaneous game where the combination of multiple choices over local users will be strategically chosen to ensure that no harm is done to anyone in terms of increasing the average prediction loss or reducing the average prediction accuracy.

# Chapter 9

# Strategically-Robust Multi-Agent Trajectory Prediction in Non-Cooperative Mobility Networks

## 9.1 Chapter Introduction

In the previous chapter, we introduced a decentralized social-aware trajectory prediction using federated learning. We personalized the model for each cluster of IID users based on user mobility features, computational resources, and network throughput. Our approach considered cooperative behavior within the collaborative clusters by sharing pre-trained federated models with all the cluster mates. However, we did not perform inter-cluster non-cooperative social learning, where users with different data features and conflicting objectives would participate in a collaborative training. Therefore, in this chapter, we aim to address the non-cooperative dynamics of a multi-agent scenario by incorporating multi-RL game-theoretic trajectory prediction.

Despite advancements in social-aware trajectory prediction, there remains a major challenge in ignoring non-cooperative neural network requirements. Existing social-aware predictors face limitations in handling non-cooperative social behaviors. These predictors are mainly centralized models where decision-making power is taken away from individuals and given to a centralized unit to make cooperative decisions by aggregating everyone's data through a single neural architecture model, as discussed in Chapter 7. In reality, humans may behave differently due to their personal goals, making it challenging for mobility predictors to accurately forecast their future actions [73]. In a non-cooperative setting, local users must have individualized architectures that suit their unique data characteristics but does not degrade other local users' performance. A specific NN architecture may be well-suited to capturing the unique data characteristics of one user, while severely degrading the prediction performance of other users due to incompatibility with their data features. Mobile users may have different mobility characteristics and objectives, which need to be considered when designing multiple competing personalized neural network architectures. Furthermore, heuristic models struggle to capture complex interactions, leading to unreliable predictions. Thus, personalizing NN architectures becomes a crucial non-cooperative

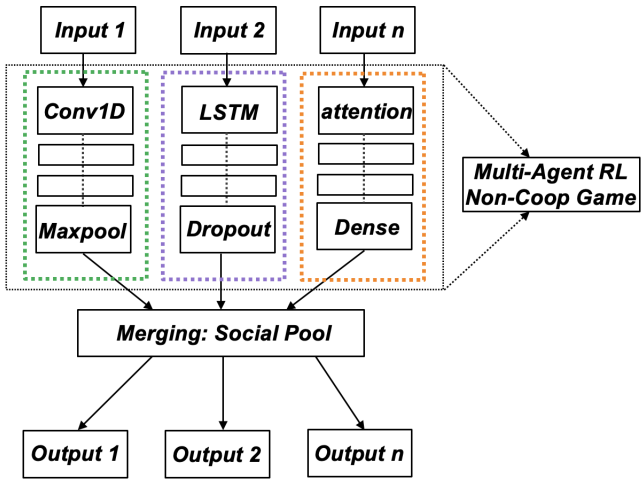



FIGURE 9.1: An overview of Multi-Agent Reinforcement Learning-designed Social Learning for non-cooperative trajectory prediction.

task in accurately predicting the joint trajectories of multiple users in a multi-agent setting. Although there are few federated learning-based decentralized trajectory predictors in literature, as discussed in Chapter 8, classical FL's limitations include the inability to personalize NN architectures as local users must have identical architectures to enable matrix summation. Therefore, the issue of neural architecture inflexibility arises in FL. Hence, a research gap remains in addressing social-aware, non-cooperative, and non-centralized trajectory prediction models. Thus, we propose a novel distributed NAS approach using multiple RLs to handle both intra-cluster requirements, where the model can be shared among cooperative users, and inter-cluster dynamics, where RLs compete to perform non-cooperative social trajectory prediction.

In this chapter, we aim to address the research question stated in Section 1.2.5 as follows.

*"RQ 5: How can we create a resilient multi-agent trajectory predictor empowered by NAS that effectively captures the dynamics of cooperative and non-cooperative user behaviors with high accuracy and efficiency?"*.

Our objective is to address the aforementioned limitations by developing a system that trains social transformers to predict the joint trajectories of multiple competing users. To achieve this, we aim to personalize the neural network architecture of each user based on their unique interests, while also taking into account the strategies employed by other users. This approach ensures that the desires and actions of one user do not negatively impact the models of other players, fostering a harmonious and cooperative environment. In pursuit of these goals, we introduce a novel model: Game-Theoretic Trajectory Prediction through Distributed Reinforcement Learning (GTP-Force) system [19][1].

[1]Partially reproduced in this chapter – Copyright ©2011 IEEE.

In response to RQ 5, there is a pressing need to develop a multi-agent trajectory prediction model capable of accurately capturing the dynamics of both cooperative and non-cooperative user behaviors. To address this requirement, our first contribution in this chapter focuses on clustering users with similar trajectories into distinct clusters. Subsequently, we consider the users within each cluster (intra-cluster) as cooperative users, enabling them to share a single neural network. Conversely, the inter-cluster users are treated as non-cooperative, and each group is assigned a separate neural architecture. To facilitate this approach, we introduce a game among the inter-cluster users, where they actively compete with one another.

In this direction, it is essential to incorporate RL-based NAS into the system in order to select an optimal combination of competing neural networks. This integration aims to optimize individual players' decision-making while simultaneously enhancing the overall strategy of the system.

Consequently, the second contribution of this work involves proposing multi-RL non-cooperative learning, where dedicated RL agents are assigned to design the best neural architectures for competing users within a multi-agent scenario. As depicted in Figure 9.1, contrasting the approach in Figure 7.2, the computing users from different clusters have their own exclusive RL agents to design their respective neural architectures. When multiple RL agents design multiple neural networks, and these networks are subsequently merged into a social pool, the models can impact the training of other users. The non-cooperative game framework aids in identifying the most effective combination of multiple neural networks for inter-cluster players.

To begin, GTP-Force clusters mobile users with similar trajectories resulting in inter-cluster users with distinct trajectories and competing mobility features. These inter-cluster users are then modeled as players in a non-cooperative game, with strategic choices made to ensure that no negative impact is placed on any individual user's decision.

To personalize the transformer's NN architectures for each non-cooperative player, GTP-Force employs an RL agent that is tailored to the specific mobility data features of the player, representative of the cluster of similar trajectories. This allows intra-cluster users with similar mobility features to be modeled using a shared NN, while inter-cluster users with different mobility features are treated as competitive players in a non-cooperative game.

To be more clear, the decentralized nature of GTP-Force allows users to keep their raw data locally, increasing privacy while decreasing communication overhead. By combining distributed reinforcement learning and game theory, GTP-Force addresses the challenge of non-cooperative behaviors in trajectory prediction, while maintaining privacy and reducing communication costs.

After the set of potentially highest-performance RL-designed NN architectures of each player is transferred to the centralized server, a social pooling layer is added to form a social-aware trajectory predictor that captures the interactions of multiple interactive players, similar to the centralized social-aware INTRAFORCE predictor presented in Chapter 7. GTP-Force's centralized server then trains different-architecture social predictors by combining users' different decisions (RL-designed neural architectures) to form a payoff matrix for a non-cooperative game.

By combining individualized NN architectures and a payoff matrix, GTP-Force offers a decentralized approach to social-aware trajectory prediction, allowing users to keep their raw data locally and increasing privacy while decreasing communication overhead. The goal of GTP-Force, is to predict the future trajectory for each user $u$, based on a user's past mobility data and inter-cluster users' mutual influences in a non-cooperative environment.

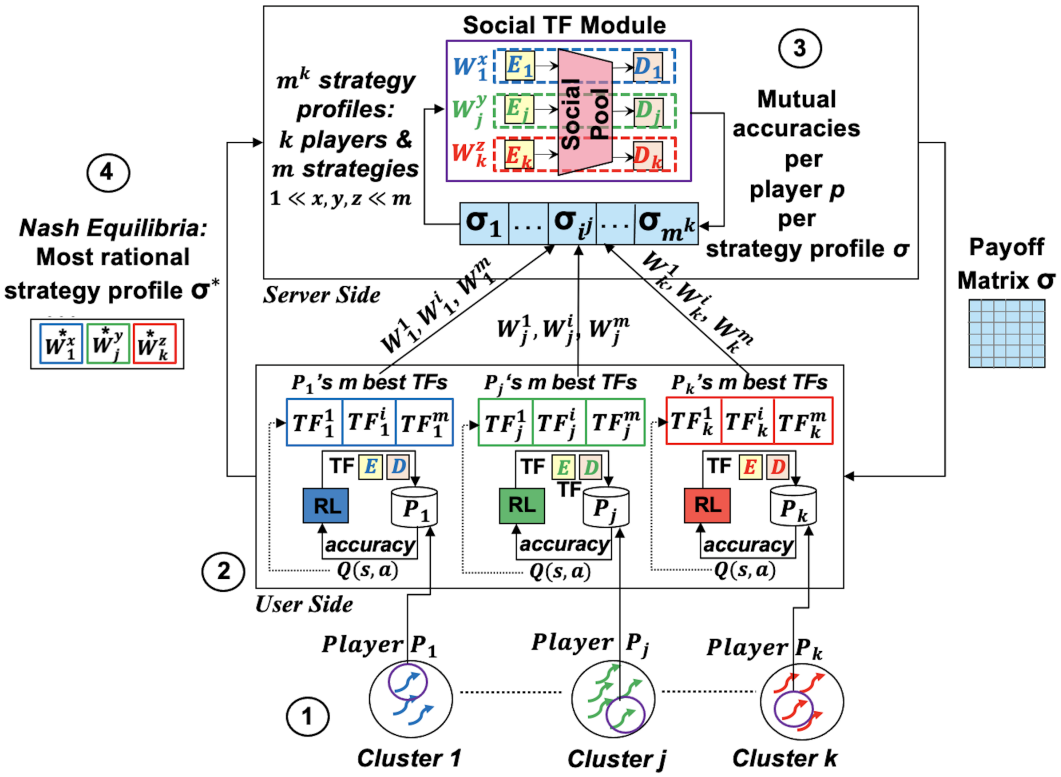



FIGURE 9.2: GTP-Force Architecture [19] ©2023 IEEE.

## 9.2 GTP-Force Trajectory Prediction System Architecture

GTP-Force workflow is divided into four phases: (1) the *game player selection*, (2) the *distributed reinforced transformer training*, (3) the *social interaction payoff computing*, (4) and the *inter-cluster non-cooperative TP training*, which are illustrated in Figure 9.2 and elaborated on below. Figure 9.2 shows a social trajectory predictor that trains on data from multiple non-cooperative players and jointly predicts multiple trajectories that are influenced by each other's mobility. Each player is selected from a distinct cluster that has unique mobility data features. It can be observed that an RL agent is used to design the transofrmer's encoder (E) and decoder (D) architectures for each of the contesting inter-cluster players. Figure 9.3 illustrates the generic architecture of a transformer, denoted as *TF(E,D)*, for an individual user. In contrast, Figure 9.2 shows how multiple *TF(E,D)s* of different individuals are merged through a *social pool* unit to form a social-aware trajectory predictor. The RL design of each player's encoder and decoder architecture must also satisfy the Nash Equilibrium of a non-cooperative game being played. This means that an RL-designed transformer for a player can only be chosen if it does not degrade the prediction output of other players in a multi-agent scenario.

Algorithm 7 details the GTP-Force system's workflow. The first section of the algorithm (lines 1 to 5) describes how competitive clients are chosen to be the players of the non-cooperative game.

**Algorithm 7:** GTP-Force Workflow [19] ©2023 IEEE

**Input:** Set of trajectories $\Theta$

**Output:** A social-TF $F_k^*$ with a combination of RL-designed NNs from Nash equilibrium

// Clustering similar trajectory users $s_{ij}$

**foreach** $(T_i, T_j) \in \Theta^2$ **do**

  $C \leftarrow \texttt{Clustering}(s_{ij})$ ;

// Build a social transformer $F_k$, $\forall c_k \in C$

**foreach** $c_k \in C$ // 1. Game Player Selection

**do**

  Elect the representative user $p_k$ for cluster $c_k$ based on Regularity Ratio;

  // 2. Distributed RL-TF Training

  // Initialize RL agent $A_k$ to optimize the TF architecture using data of player $p_k$

  $A_k \leftarrow \texttt{InitAgentRL}(\gamma, \alpha, \varepsilon, \varepsilon_0)$

  // Initialize state-action table to zero for all states and actions

  $\forall (s, a) \in S \times A : Q(s, a) \leftarrow 0$;

  // Initialize exploration probability $\varepsilon$ to maximum and empty architecture state

  $\varepsilon \leftarrow 1, s \leftarrow \emptyset$;

  // Optimize TF architecture up to $v_{\max}$ episodes

  **foreach** $v \in \{1, \ldots, v_{max}\}$ **do**

    // Decrease exploration every $v_{\max}\varepsilon_0$ episodes

    **if** $v \mod v_{max}\varepsilon_0 = 0$ **then**

      $\varepsilon \leftarrow \varepsilon - \varepsilon_0$;

    // $\varepsilon$-greedy to select next TF architecture

    **if** $\texttt{RandomSample}([0, 1]) \leq \varepsilon$ **then**

      $a_v \leftarrow$ random action $a \in A(s)$;

    **else**

      $a_v \leftarrow \arg\max_{a \in A(s)} Q(s, a)$;

    // Update state according to action $a_v$

    $s' \leftarrow \texttt{UpdateState}(s, a_v)$;

    // Train the TF with data of the representative user $p_k$ for a few epochs $\theta_s$. Compute model error $w$ and reward $\rho_v$ of architecture modification (action $a$)

    $s'_* \leftarrow \texttt{Train}(s', r_k, \theta_s)$;

    $w \leftarrow \texttt{ComputeModelError}(s'_*, r_k)$;

    $\rho_v \leftarrow 1/w$;

    // Update state-action table

    $Q(s, a_v) \leftarrow (1 - \alpha) Q(s, a_v) + \alpha \left( \rho_v + \gamma \max_{a \in A(s')} Q(s', a) \right)$;

    $s \leftarrow s'$;

  // Select $m$ trained NN weights with lowest loss

  $W_k \leftarrow \arg\max_{W' \subseteq \{w_1, \ldots, w_m\}} Q(s, \cdot)$ ;

// 3. Social Interaction Payoff Computing

**foreach** $\sigma \in \{1, \ldots, m^k\}$ **do**

  // Train the Social-TF architecture with $k$ Encoder Stacks connected to $k$ Decoder Stacks through one social pool for $k$ players

  $F_k \leftarrow \texttt{Train}(f_k, c_k, \theta_s)$;

// Form the Payoff Matrix A from strategy $\sigma_k$ and prediction accuracy $F_k$

$A \leftarrow (\sigma_k, F_k), \forall k \in K$;

// 4. Non-Cooperative Trajectory Prediction

// Compute Nash Equilibria through finding the best response of player $j \in K$

$b_j(\sigma_{-j}) \leftarrow \arg\max_{\sigma_j \in \Sigma} u_j(\sigma_j, \sigma_{-j})$ ;

$F_k^* \leftarrow \sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_k^*)$;

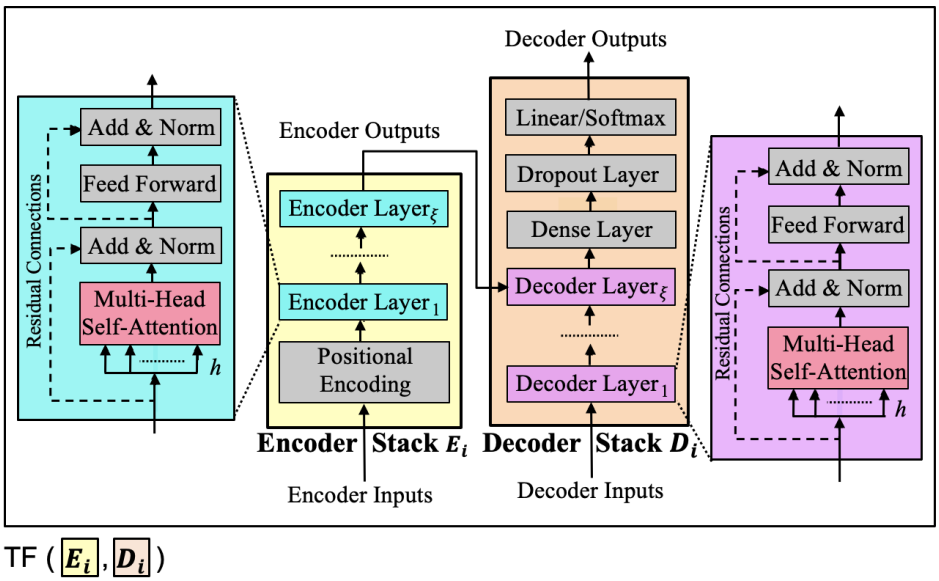



FIGURE 9.3: Transformer's Encoder-Decoder TF(E,D) block for an individual user [20] ©2022 IEEE.

In this direction, the algorithm forms disjoint clusters of similar-trajectory users. Afterward, the algorithm assumes that intra-cluster users show cooperative behavior, while, inter-cluster users are each others' contestants with non-cooperative attitude. Therefore, from each cluster one user is chosen as the player of that cluster. The second section of the algorithm (lines 6 to 22) describes how players train locally the RL-designed transformers tailored to their unique characteristics of their clusters. For each player, the RL agent provides a list of top high-performance transformer neural networks. The third section of the algorithm (lines 23 to 25) describes how social interaction among players is extracted for different combinations of their strategies (transformers) to form the payoff matrix. Finally, the fourth section of the algorithm (lines 25 to 27) explains how the non-cooperative game can be efficiently played by satisfying the Nash Equilibrium. In this section, the algorithm aims to discover the optimal combination of diverse players' neural architecture choices for training a social predictor. The goal is to ensure that no player's architecture significantly hampers the performance of other players. Once the optimal strategy profile has been identified, the joint trajectory prediction can be performed.

### 9.2.1 Game Player Selection

We assume that the $n$ users in the system can be classified into $k$ clusters of users with similar trajectories, where $k << n$. Users are partitioned into distinct clusters $C = c_i \subseteq \Theta | i = 1, \ldots, q$, where $c_i \cap c_j = \emptyset$ for all $i, j \in 1, \ldots, q$, $i \neq j$. Users within a cluster have comparable mobility features and can be modeled using a single TF NN model. In contrast, inter-cluster users have distinct mobility features and are treated as competitors, each with their own preferences and

desirable strategies. To model the non-cooperative game among inter-cluster users, we select one player from each cluster. The chosen player user of cluster $c_k$ is denoted as $p_k$.

For selecting a *player* from each cluster, we prioritize those with the most reliable and periodic data quality. This guarantees that the users that are chosen to play in the game can design the best NN for the entire cluster. However, since GTP-Force is a distributed system and the central server lacks access to local users' raw data, local users themselves can calculate their *regularity ratios* in both the time and frequency domains using our proposed regularity ratio, as presented in Section 4.6. This metric allows local users to estimate the quality and periodicity of their own data and only transmit regularity ratio values to the centralized server. In turn, this enables the central server to select a player from each cluster with the most reliable estimated data. The acceptable threshold of regularity ratios in both domains is set empirically based on the specific dataset being used. By combining information from both domains, we can achieve a decent estimation of the quality and periodicity of the local user data. The central server selects a user as the player for the game only if their regularity ratios meet the threshold criteria in both time and frequency domains.

Limiting the engagement to only one player per cluster of similar users has two major benefits: First, it reduces the computational and communication overheads associated with personalizing the neural network architecture for each user, and secondly, it simplifies the complexity of the game and facilitates the identification of the Nash equilibrium. As the number of players and possible decisions increase, the size of the payoff matrix grows exponentially, making it more challenging to identify a dominant strategy and Nash equilibrium. By limiting the engagement to one player per cluster, the model personalization and strategic decision-making can scale effectively in large-scale non-cooperative networks.

After the non-cooperative training phase, GTP-Force efficiently distributes the pre-trained models to all users of each cluster from their respective cluster player users. This approach ensures that every user in the system can perform prediction tasks effectively, without compromising the accuracy or computational complexity of the model. This method is more efficient than using pruning algorithms to manage the complexity of computations, as it eliminates the need for complex pruning techniques and enables individual users to benefit from the same high-performing models.

### 9.2.2 Distributed Reinforced Transformer Training

During the personalized transformer architecture training phase, each player in GTP-Force trains an RL agent on their local data to personalize the transformer NN architecture based on their unique mobility features, or in other words, intra-cluster features. Meanwhile, inter-cluster users play in a non-cooperative game.

GTP-Force uses $\varepsilon$-greedy $Q$-learning policy for each RL agent to design a list of $m$ optimal transformer NN architectures. These architectures are then transferred to the server to be played in a non-cooperative game so that the best NN per user can be chosen through Nash.

The action space for each RL agent, which personalizes each player, is the hyperparameters of the transformers, as presented in Figure 9.3. The action space for the GT agent is the set of all possible combinations of multiple players' different transformer architectures, which is interpreted as different strategies of the game.

During the exploration phase, at each episode of local RL, the agent proposes a transformer NN architecture, and the unknown reward associated with the selected architecture is evaluated by training the suggested transformer on the player's data. On each RL episode $t$, the NN architecture proposed by the agent achieves a loss $L_t \in \mathbb{R}$, which is defined as Sparse Classification Cross-Entropy for classification and Mean Square Error for regression. The RL agent uses a reward function $r_t = -L_t$ at each episode $t$ to evaluate the performance of the candidate NN architecture and updates its Q-table, similar to the equation shown in Chapter 8.

Finally, the RL agent completes its training on the chosen architecture until the model converges over the exploitation phase. At the end, each RL agent selects the $m$ best NN architectures by searching the $m$ states $s^*$ corresponding to the $m$ maximum values of the Q-table shown in Equation 8.2.

The goal of GTP-Force is training multiple RL agents on multiple decentralized players to design multiple transformer NN architectures in a non-cooperative multi-agent environment that minimize the model losses based on the features of different players. This approach allows for personalized model training while still ensuring scalability and efficient decision-making.

### 9.2.3 Inter-Cluster Social Interaction Payoff Computing

At this point, each player's trained RL agent sends a list of its $m$ best NNs' trained weights as the its $m$ possible decisions or strategies to the central server for the game. The central server contains a social-pool module where multiple users transformer NN encoders and decoders can be aggregated forming a social model. Therefore the joint predicted trajectories can show the output of each players decision given other players' strategies. We can model the social trajectory prediction through a simultaneous non-cooperative game where the goal of GTP-Force is to find the best combination of NNs for different conflicting players so that the interest of each player is optimized while taking into account other users strategies.

In a classic non-cooperative simultaneous game, each player chooses their strategy independently of the other players, and there is no communication or coordination between players. Multiple players make decisions simultaneously, without knowing the decisions made by the other players. Let $K = \{p_1, \ldots, p_k\}$ be the set of players, where each player $p_j \in K$ can select a strategy $\sigma_j \in \Sigma$ among $m = |\Sigma|$ possible strategies. The payoff for each player depends on the combination of strategies chosen by all players. The game's outcomes for each player can be represented by a payoff matrix $A \in \mathbb{R}^{m^k \times k}$ with entries $a_{\sigma_1,\ldots,\sigma_k,j}$, where $\sigma_1, \ldots, \sigma_k \in \Sigma$. Each row of matrix $A$ represents one of the $m^k$ possible combinations of strategies for all players [65]. The entry $a_{1,\ldots,k,j}$ represents the payoff for player $j$ when the players choose the strategies $\sigma_1, \ldots, \sigma_k$. The matrix $A$ can be constructed as follows:

$$A = \begin{bmatrix} a_{1,\ldots,1,1} & a_{1,\ldots,1,2} & \cdots & a_{1,\ldots,1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,\ldots,m,1} & a_{m,\ldots,m,2} & \cdots & a_{m,\ldots,m,m} \end{bmatrix} \tag{9.1}$$

### 9.2.4 Non-Cooperative Mobility User Training

During this phase, the central server sends the completed payoff matrix to all players. Each element of the matrix contains $k$ joint predictions for $k$ players, revealing the impact of each player's decision strategy (chosen NN architecture) on others in terms of social TP accuracy.

In the analysis of the non-cooperative simultaneous game, the concept of Nash equilibrium is of utmost importance for determining the best strategy (in our case, best combination of NN architectures) for multiple players.

The Nash equilibrium signifies a set of strategies in which no player is motivated to alter their strategy, provided they are aware of the strategies of the other players. Nash equilibrium is not necessarily the combination that yields the maximum total or expected payoff, but rather a set of strategies that is rational, stable, and self-enforcing, given the strategies chosen by other players. As the players act independently and may not have knowledge of other players' strategies, it may not always be possible to select the optimal solution.

A strategy profile $\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_k^*)$ is a Nash equilibrium if, for each player $j \in K$, their strategy $\sigma_j^*$ is the best response $b_j$ to the strategies of the other players, i.e., $\forall \sigma \in \Sigma^k : u_j(\sigma^*) \geq u_j(\sigma)$. In this context, $u_j$ denotes the utility function of player $j$, which represents the player's preference over the possible outcomes. In other words, player $j$'s strategy $\sigma_j^*$ is a best response to the strategies $(\sigma_1^* \sigma_2^* \ldots, \sigma_{j-1}^* \sigma_{j+1}^* \ldots, \sigma_k^*)$ chosen by the other players. To find the best response of player $j$, $b_j$, in a non-cooperative game, we use the following formula:

$$b_j(\sigma_{-j}) = \arg\max_{\sigma_j \in \Sigma} u_j(\sigma_j, \sigma_{-j}), \tag{9.2}$$

where $\sigma_{-j}$ denotes the strategies of all other players in the game. To determine whether a given strategy profile is a Nash equilibrium, we can find the best response of each player to the strategies of the other players, using the formula for the best response given above, substituting $\sigma_{-j}$ with $(\sigma_1^*, \sigma_2^*, \ldots, \sigma_{j-1}^*, \sigma_{j+1}^*, \ldots, \sigma_k^*)$. If each player's chosen strategy is the best response to the strategies of the other players in the given strategy profile, then the strategy profile is a Nash equilibrium [65].

Given that players' RL agents are rational and using the Nash equilibrium to personalize their NNs, GTP-Force proceeds to evaluate the performance of the social-aware TP by selecting the best combination of decisions in a multi-agent scenario.

## 9.3 Evaluations

We evaluate the performance of GTP-Force against several existing mobility prediction models in two distinct scenarios. The first scenario involves a small moving area for users, while the second scenario deals with a larger moving area, commonly referred to as the *small-scale* and *large-scale* mobility scenarios, respectively. As discussed in Section 4.4, the large-scale scenario is based on the *Orange* telecommunication S.A. [104] private dataset, while the small-scale scenario is based on the *ETH* [68] plus *UCY* [47] public camera dataset.

The constant parameters used to train the Social TP model, TF predictor, and RL agent used through GTP-Force are shown in the first two sections of Table 9.1. Additionally, the third section of Table 9.1 displays the search space of hyperparameters that form the TF architecture, with each row corresponding to one of the RL's potential actions.

After clustering similar trajectory users and reducing the number of players in the non-cooperative game, we are able to solve the Nash equilibrium using a brute force approach. However, it should be noted that finding Nash equilibrium is a computationally challenging task and the brute force approach may not be practical for large games involving many players. In such scenarios, we consider using more advanced algorithms or heuristics, such as the Lemke-Howson algorithm or support enumeration algorithm, which can offer more efficient solutions for finding Nash equilibrium.

### 9.3.1 Large-scale Mobility Scenario

For evaluating the GTP-Force performance in predicting future base station IDs in the *large-scale* scenario, we conduct two experiments. In the first experiment, we compare the prediction accuracy and build time for individual user TP of RL-TF with several other trajectory predictors, including NN-based predictors RL-CNN, RL-LSTM, HO-LSTM, and GS-LSTM, as well as non-NN predictors J48 Decision Tree, and XGBoost and RF ensemble models. RL-LSTM uses RL to optimize LSTM architecture, while HO-LSTM and GS-LSTM use Hyperopt and Grid Search, respectively. The aim of this experiment is to demonstrate that TFs and RL outperform other ML predictors and hyperparameter optimization models.

In the second experiment, we evaluate the performance of the non-cooperative strategic GTP-Force, using GT and RL on TF NNs, as a social-aware TP model in terms of accuracy and model size performance metrics, in comparison to individual RL-designed TF (RL-TF), classical GT with TF NNs without RL personalization (GT-TF), and the classical social-aware model with TF NNs without strategic GT and RL personalization (Social-TF).

### 9.3.2 Small-Scale Mobility Scenario

To assess the performance of GTP-Force in predicting future location coordinates of users in the *small-scale* scenario, we compared it with several group intelligence-based social trajectory predictors. These include Social-LSTM [3], Social-GAN [31], Sophie [74], Social-Ways [4], Social-STGCNN [60], STAR [94], INTRAFORCE [20], and ATPFL [87], based on their average displacement error.

### 9.3.3 Large-Scale Evaluation Results

The results obtained from the large-scale experiment conducted (Orange dataset) reveal that utilizing RL to explore the NN architecture of the TF, and GT to model the decision-making of non-cooperative multi-agent NN designs leads to superior prediction accuracy, expedited model building time, and reduced training parameter size compared to other trajectory prediction methods.

In Figure 9.4, we compare the Kernel Density Estimation (KDE) of accuracy between RL-TF and other predictors. KDE is a non-parametric method used to estimate the probability density

TABLE 9.1: Experimental parameters of GTP-Force for small-scale and large-scale scenarios) [19] ©2023 IEEE

| Transformer Parameters | |
|---|---|
| Batch size (small-scale, large-scale) | 10, 200 |
| Learning rate decay | 0.002 |
| Social Transformer training epochs $\theta_l$ | 200 |
| Early stopping patience (in epochs) | 10 |
| Early stopping threshold (small-scale, large-scale) | 0.05, 0.1 |
| Dense layers' activation func. (hidden, output) | ReLU, SoftMax |
| **Reinforcement Learning Parameters** | |
| Maximum RL training episodes $v_{\max}$ | 500 |
| Training epochs per episode $\theta_s$ | 20 |
| Discount factor $\gamma$, learning rate $\alpha$ | 1, 0.01 |
| Exploration rate decay $\varepsilon_0$ | 0.1 |
| Training target per episode (small-scale) $\eta$ | 0.05 |
| Training target per episode (large-scale) $\eta$ | 0.1 |
| Exploration training validation (small-scale) | 4 sets train, 1 set test |
| Exploration training validation (large-scale) | 10-fold x-validation, 70% train, 30% test |
| **RL Agent Actions: Transformer Hyperparameters Space** | |
| Number of hidden layers | $10, 11, \ldots, 50$ |
| Number $\xi$ of encoder and decoder layers | $1, 2, 3, 4, 5$ |
| Number of heads $h$ in a multi-head attention layer | $2, 4, 6, 8$ |
| Dimension of the key for a self-attention layer | $64, 128, 265$ |
| Normalization layer parameter | $10^{-2}, 10^{-3}, 10^{-6}$ |
| Number of perceptrons in dense layer | 20, 50, 80, 100, 150 |
| Dropout ratio in dropout layer | 0.15, 0.25, 0.5, 0.75 |

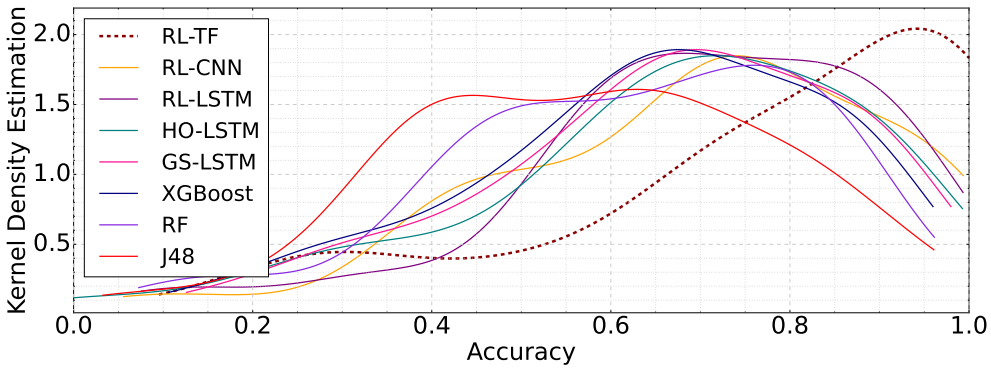



FIGURE 9.4: Accuracy KDE of different trajectory predictors trained on an individual user data for the large-scale scenario (Orange dataset) [19] ©2023 IEEE.

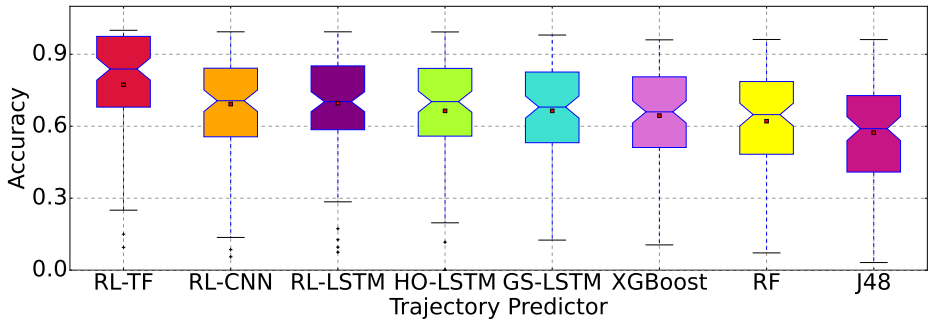



FIGURE 9.5: Accuracy of different trajectory predictors trained on an individual user data for the large-scale scenario (Orange dataset) [18, 19] ©2023 IEEE.

function of a random variable. It can be observed that the distribution of user prediction accuracy for RL-TF is skewed towards higher values, indicating that it is better at predicting user trajectories with greater precision.

We observe that in Figure 9.5 RL-TF archives mean accuracy of (75%), which is 10% greater than the other reinforced models (RL-LSTM and RL-CNN) and almost 20% higher than non-NN models (XGBoost, RF, and J48). Achieving a higher prediction accuracy over the Orange dataset is restricted by the limited dataset size (63 days) and the significant diversity in users' data sample distributions. The obtained accuracy of 75% is the average accuracy over 100 random users with highly variable data quality and periodicity.

In Figure 9.6, it is evident that RL-TF demands slightly more than one hour of build time, which is comparable to RL-CNN and RF. Despite having larger architectures than CNNs and LSTMs, TFs' attention mechanism notably diminishes the build time. Furthermore, the RL-TF build time is analogous to that of RF, which does not necessitate architecture exploration but only training. However, RL-LSTM and HO-LSTM involve extended build times due to the sequential nature of LSTMs and the extensive training of HO exploration.

In Figures 9.7a and b, we demonstrate the impact of tuning the number of dense layers and TF

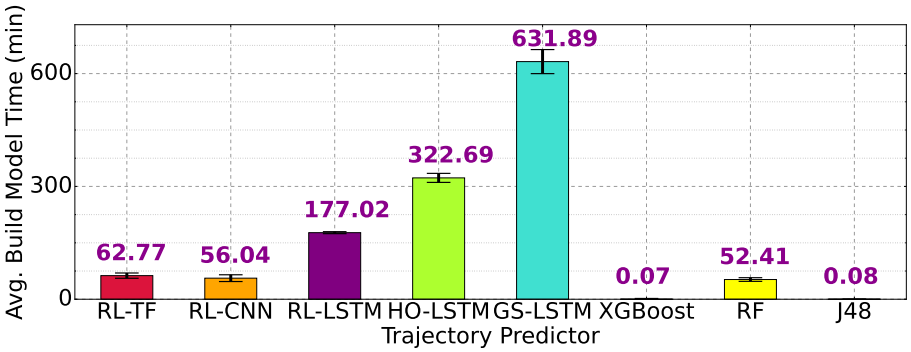



FIGURE 9.6: Build time of different trajectory predictors trained on individual user data for the large-scale scenario (Orange dataset) [18, 19] ©2023 IEEE.

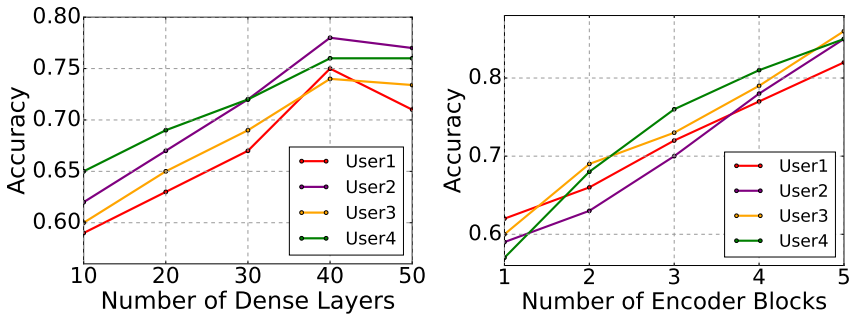



FIGURE 9.7: Dense layer (a) and encoder (b) hyperparameters studying for RL-TF.

encoder block hyperparameters on the prediction accuracy of four random users. This provides an insight into how the RL agent explores various hyperparameters to discover high-performance NN architectures. Of particular interest is the observation in Figure 9.7a, which indicates that as the number of dense layers increases up to a certain point, the accuracy also increases. However, after this point, the accuracy either stays constant or decreases. This illustrates that overfitting occurs when the model learns to fit the noise in the training data, rather than the underlying patterns. As a result, the model exhibits poor generalization performance on new, unseen data. Due to the heavy computational requirements of TF encoders, the RL agent attempts up to five encoder blocks, as illustrated in Figure 9.7b. This figure reveals that as the number of encoder blocks increases, the accuracy also increases, taking into account that the computational demands also rise considerably.

To have fair evaluations, we compare the average accuracy of GTP-Force, which employs non-cooperative GT in social TP via RL-designed TFs, with respect to RL-designed Social-TF, Game-theoretic Social-TF without RL personalization, and Social-TF without RL personalization or non-cooperative GT modeling, as shown in Figure 9.8. It can be observed that GTP-Force achieves an average accuracy of 80%. This represents a 5% improvement over the RL-based TF predictor,

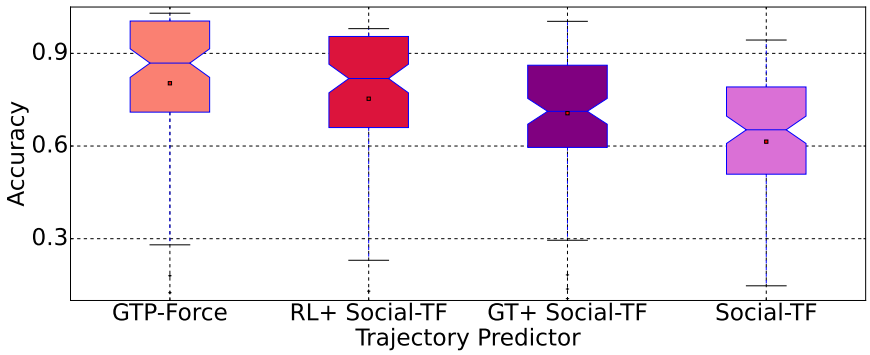



FIGURE 9.8: Accuracy of GTP-Force with respect to RL-based TF, GT-based TF without RL, and social-TF in the large-scale scenario (Orange dataset) [19] ©2023 IEEE.

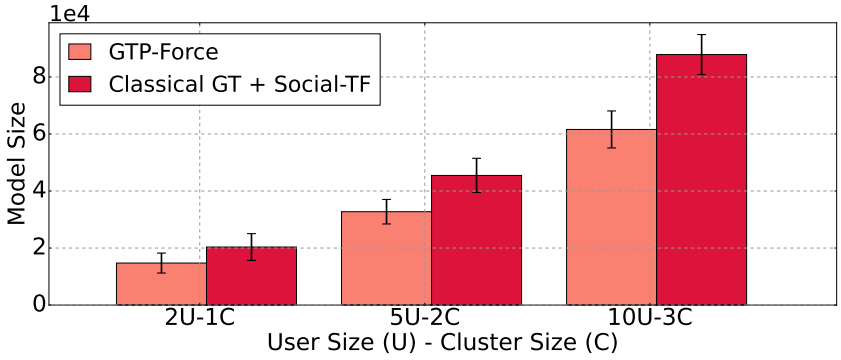



FIGURE 9.9: Model size of the GTP-Force with respect to the classical GT for the social-TF trajectory prediction in the large-scale scenario (Orange dataset) [19] ©2023 IEEE.

a 10% improvement over the classical GT-based TF predictor without RL personalization, and a 15% improvement over the simple Social-TF without RL or GT.

Figure 9.9 compares the average model size of GTP-Force, an RL- and cluster-based game theoretic approach, with that of the classical GT approach for social-aware transformer-based trajectory prediction across various numbers of users. The model size difference gap between GTP-Force and the classical GT approach increase as the number of users, and subsequently the number of clusters, rises. Despite the expected increase in complexity due to the use of RL, GTP-Force clusters similar users and selects a small number of players from each cluster for the game, resulting in a smaller neural model size than the classical approach that is played and trained among a greater number of players whose neural architectures are not even optimized.

TABLE 9.2: ADE [m] of different social trajectory predictors for the small-scale scenario (ETH+UCY datasets) [19] ©2023 IEEE

| Work | ETH | Hotel | Univ | Zara1 | Zara2 | **Mean** |
|---|---|---|---|---|---|---|
| Social-LSTM | 1.09 | 0.79 | 0.67 | 0.47 | 0.56 | **0.72** |
| Social-GAN | 0.81 | 0.72 | 0.60 | 0.34 | 0.42 | **0.58** |
| SoPhie | 0.70 | 0.76 | 0.54 | 0.30 | 0.38 | **0.54** |
| Social-BiGAT | 0.69 | 0.49 | 0.55 | 0.30 | 0.36 | **0.48** |
| Social-Ways | 0.39 | 0.39 | 0.55 | 0.44 | 0.51 | **0.46** |
| Social-STGCNN | 0.64 | 0.49 | 0.44 | 0.34 | 0.30 | **0.44** |
| PECNet | 0.54 | 0.18 | 0.35 | 0.22 | 0.17 | **0.29** |
| STAR | 0.36 | 0.17 | 0.31 | 0.26 | 0.22 | **0.26** |
| INTRAFORCE | 0.31 | 0.24 | 0.22 | 0.14 | 0.23 | **0.22** |
| **GTP-Force** | **0.27** | **0.17** | **0.21** | **0.16** | **0.18** | **0.19** |

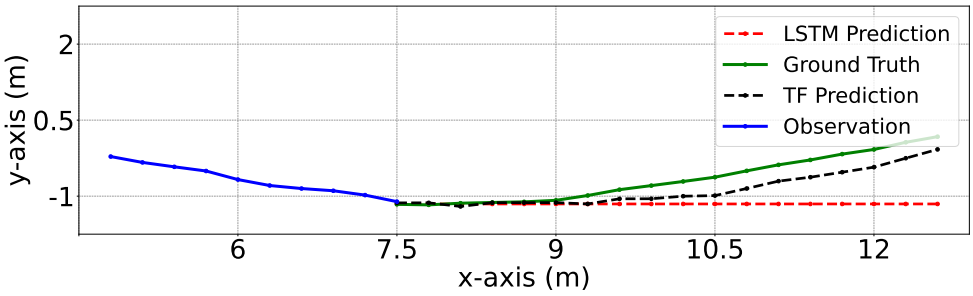



FIGURE 9.10: Predicted trajectories by TF and LSTM versus the ground-truth trajectory [19] ©2023 IEEE.

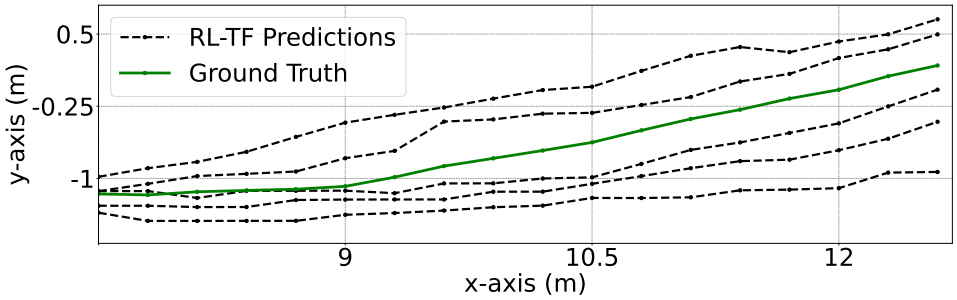



FIGURE 9.11: Various RL-designed TF predictions versus the ground-truth path in the small-scale scenario (ETH+UCY datasets) [19] ©2023 IEEE.

### 9.3.4 Small-Scale Evaluation Results

Table 9.2 shows the results of the small-scale experiment (ETH+UCY datasets), in which GTP-Force achieves the lowest ADE (0.19m) compared to several state-of-the-art social trajectory predictors.

Figure 9.10 displays several predicted trajectory points by both TF and LSTM, as compared to the ground-truth trajectory. We can observe that the optimal RL-designed TF path more closely resembles the ground-truth path than the optimal RL-designed LSTM path.

Figure 9.11 depicts various RL-designed TF predictions, corresponding to different TF NNs, as compared to the ground-truth path. We can discern that RL agent's different random actions (proposed TF NNs) result in different prediction accuracies. Moreover, in this figure, we show that how we sort each player's $m$ best actions based on their deviation from the ground-truth path to define its decision choices for the game.

## 9.4 Chapter Conclusions

We introduce GTP-Force, a novel system for constructing a highly accurate trajectory predictor that captures the social interactions among multiple mobile users. Utilizing Reinforcement Learning, GTP-Force designs the high-performance Transformer architecture based on the *intra-cluster* user mobility features. Subsequently, GTP-Force applies a non-cooperative game theory to determine the optimal combination of *inter-cluster* neural network architectures in the dynamic multi-agent environment.

Our evaluation of GTP-Force is conducted on both small and large-scale scenarios, utilizing the ETH+UCY and Orange datasets, respectively. In the small-scale scenario, the proposed GTP-Force attains an ADE of 0.19m, which corresponds to a lower positioning error when compared to several state-of-the-art models. In a large-scale scenario, our study demonstrates that Reinforced Transformers outperform both of the best baselines, LSTM- and CNN-based predictors, by achieving up to 10% higher accuracy and reducing training time by up to 70%. Our trajectory predictor, based on the Reinforced Transformer in game theory, achieves 80% average accuracy, 5% higher than RL-designed Social-TF, 10% higher than game theoric Social-TF (without RL personalization), and 15% higher than the simple Social-TF (without RL and GT). Our experiments show that by increasing the number of users in the system and subsequently in each cluster of similar behavior mobile users, GT-Force significantly reduces the number of training parameters compared to the classical game theoretic Social-TF so that the model size gap between GTP-Force and the classical GT approach widens, with GTP-Force consistently achieving a smaller model size for trajectory prediction.This result suggests that our proposed approach is highly scalable and efficient, making it suitable for large-scale scenarios while maintaining high prediction accuracy.

# Chapter 10

# Conclusions

## 10.1 summary

The increasing prevalence of mobile devices and the Internet of Things has led to the widespread availability of mobility and location data. This data provides numerous benefits, including the ability to predict human, vehicle, and robot trajectories. These predictions have a wide range of applications, such as in intelligent transportation, urban planning, rescue and emergency situations, autonomous vehicle driving, collision avoidance, and improving the quality of service for modern wireless network services like handover management and service migration.

Trajectory prediction involves forecasting the path an individual or a group may take at a specific time and location, based on their previous behavior and mutual interactions within the complex urban environment. Mobility and location data, such as GPS data, sensor data, IoT devices' data, social platforms location tags, WiFi access point data, and visited base station data, provide valuable inputs for trajectory prediction models. Machine learning and deep learning algorithms can then analyze and process this data to make accurate predictions. The power of mobility and location data can provide valuable insights into human behavior, which can be used to improve critical services' planning and decision-making.

In this thesis, we have developed advanced trajectory prediction models and decision-making algorithms for various urban environment scenarios, including isolated, multi-agent, centralized, distributed, cooperative, and strategic scenarios, using state-of-the-art machine learning and deep learning techniques. The primary aim of these predictors is to reduce the loss and mean squared error of trajectory predictions while simultaneously minimizing the model size, number of training parameters, training time, and transmission time through reinforcement learning. Additionally, by clustering users with similar trajectories and tailoring the models to their intra-cluster unique data, and by considering the available wireless network bandwidth, we can improve prediction accuracy in specific regions while minimizing the computational load on the prediction system.

Furthermore, we have also introduced strategic games to model inter-cluster dynamics, where users possess distinct preferences and mobility characteristics. Through the analysis of game-theoretic solutions, we can gain insights into how user behavior impacts overall mobility patterns

and develop strategies to enhance the efficiency of predictive systems.

To evaluate the performance of our trajectory predictors, we conducted experiments on two distinct datasets: *small-scale* (image dataset) and the *large-scale* (mobile network dataset) mobility scenarios. For the large-scale scenario, we used a private dataset from Orange telecommunication S.A. [104], where users can move over a wide area spanning several kilometers. Therefore, GPS coordinates of a user within a small area are not relevant for trajectory prediction, and the trajectories are defined as a sequence of visited cellular base stations over time, making the prediction task a classification task. For the small-scale scenario, we used the ETH [68] plus UCY [47] public camera dataset, where users move within a small area of a few tens of meters. In this case, the information about the base station a user is connected to is not relevant for trajectory prediction and does not contribute to improving the task. Therefore, we only considered the small-scale user position coordinates in the trajectories, which made the prediction task a regression task. Through these experiments, we demonstrated the effectiveness of our trajectory predictors and decision-making algorithms in various scenarios with respect to several state-of-the-art mobility and trajectory predictors.

To demonstrate the advantages of our advanced trajectory prediction algorithms, we have conducted extensive testing on real-life network applications, including handover management and service migration in multiple-input and multiple-output (MIMO) systems. By accurately predicting the trajectories of mobile devices, we can optimize the handover process between base stations, reducing latency and enhancing the overall user experience. Additionally, our trajectory prediction algorithms can assist in service migration, ensuring seamless and uninterrupted connectivity for mobile users as they move between different locations. Our research has shown that the use of advanced trajectory prediction algorithms can significantly improve the efficiency and performance of real-life network applications, providing a more reliable and seamless experience for mobile users.

In the following two sections, we will reiterate the significant contributions of our work in optimizing performance metrics and highlight our achievements and conclude this thesis by highlighting potential research directions for future studies.

## 10.2 Contributions

In this section, we will provide a summary of the contributions made in this thesis by revisiting the research questions we designed and the proposed systems and methods to address them.

### 10.2.1 Trajectory Prediction-Driven Handover Management and Service Migration in Multi-Access Edge Computing Environments

In Chapter 5, we introduce three high-performance models: RL-LSTM, RL-HEC, and RL-SM. These models encompass an automated trajectory predictor, a proactive handover management model, and an anticipatory service migration model, respectively.

In this chapter, we responded to research questions RQ 1.1 and RQ 1.2 as follows:

- *RQ 1.1: What strategies can be implemented to enhance the dependability of existing neural architecture search techniques in trajectory prediction, while maintaining an optimal trade-off between neural network accuracy and convergence rate?*

  We proposed a reinforcement learning method for personalizing the architecture search for LSTM networks with high prediction accuracy and a fast convergence rate. Our reinforcement learning method offers a more efficient solution to the non-convex, non-linear NP-hard neural architecture search problem compared to existing AutoML approaches. We validated our proposed RL-LSTM approach using a large-scale real-world anonymized dataset collected from the Orange telecommunication network operator. Through extensive experimentation, we have demonstrated that our proposed RL-LSTM trajectory predictor surpasses various state-of-the-art neural network predictors and neural architecture search methods in terms of performance, training time, and accuracy.

- *RQ 1.2: How can we establish a resilient proactive handover mechanism that ensures high-quality service and uninterrupted network connectivity for applications such as service migration, leveraging the capabilities of high-performance neural networks?*

  Leveraging our proposed high-performance RL-LSTM mobility predictor, we have devised a proactive handover management and service migration algorithm tailored for MEC scenarios in modern wireless networks. Our experimental results demonstrate that our novel proactive handover algorithm and service migration scheme, which leverages mobility prediction, outperforms state-of-the-art handover management algorithms. Our simulation results show that the proposed solutions can effectively reduce ping-pong handover rates to almost zero while increasing the measured network throughput by 1.5 times compared to existing solutions. This leads to a significantly lower number of migration attempts and failures over service migration applications, improving the quality of service.

For more comprehensive insights and detailed information, we recommend reading our journal paper [104], which extensively examines the subject matter.

### 10.2.2 Large-Scale Individual-Agent Trajectory Prediction

In Chapter 6 we proposed RC-TL, an automated neural network design trajectory prediction system for large-scale network scenarios to guarantee scalability.

In this chapter, we responded to research questions RQ 2.1 and RQ 2.2 as follows:

- *RQ 2.1: What alternative neural network architectures can be employed to replace conventional RNNs in trajectory prediction, allowing for more effective capturing of complex spatiotemporal mobility features?*

  We proposed a 1D-CNN-based trajectory predictor as an alternative to the widely used LSTMs. The choice of 1D-CNNs offers distinct advantages, including enhanced speed and robustness.

- *RQ 2.2: How can we efficiently scale neural architecture search to personalize trajectory prediction for multiple individuals within large-scale networks, taking into account the challenges arising from the*

*heterogeneity of users' data? What is the optimal balance between computational resource utilization and prediction accuracy in this context?*

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

For more comprehensive insights and detailed information, we recommend reading our RC-TL paper [21], which extensively examines the subject matter.

### 10.2.3 Social-aware Multi-Agent Trajectory Prediction

In Chapter 7 we presented INTRAFORCE, a system to build a trajectory predictor that learns the social interaction within clusters of similar mobile users.

In this chapter, we responded to research questions RQ 3.1 to RQ 3.3 as follows:

- *RQ 3.1: What alternative neural network architectures can be employed to replace conventional CNNs in trajectory prediction, allowing for more effective learning of complex time-series mobility data?*

  We proposed a transformer-based trajectory predictor as an effective alternative to CNNs, leveraging the advancements in time-series analysis. While CNNs excel in image processing, transformers have emerged as specialized models for time-series data. Therefore, we expanded the utilization of transformers from neural language processing to trajectory prediction.

- *RQ 3.2: How can we effectively reduce the computational complexity of existing social-aware methods while preserving the overall accuracy of the system's trajectory predictions?*

- *RQ 3.3: How can we integrate the personalization paradigm of neural architecture search into multi-agent social trajectory prediction, enabling the search for high-performance neural networks tailored to the unique mobility patterns of interactive users, rather than focusing solely on individuals?*

  We proposed a multi-agent transformer-based trajectory predictor, called INTRAFORCE, which uses Reinforcement Learning to build a Social-Transformer architecture based on the

intra-cluster user mobility features. INTRAFORCE is evaluated on small and large-scale scenarios, based on the ETH+UCY and the Orange datasets, respectively. In the small-scale scenario, INTRAFORCE achieves an average displacement error of 0.22, which corresponds to a lower positioning error compared to several state-of-the-art models. In the large-scale scenario, we show that Reinforced Transformers outperform LSTM- and CNN-based predictors by achieving up to $+10\%$ accuracy and up to $-70\%$ training time, and outperform non-neural models based on RF and J48 of up to $+20\%$ accuracy. Our experiments show that increasing the number of users in a cluster leads to slightly higher accuracy, while considerably decreasing the time needed to build and train the trajectory predictors, as well as the number of training parameters.

For more comprehensive insights and detailed information, we recommend reading our INTRAFORCE paper [20], which extensively examines the subject matter.

### 10.2.4 Distributrd Multi-Agent Trajectory Prediction

In Chapter 8, we proposed FedForce, a distributed trajectory predictor based on a federated learning paradigm for decentralized multi-agent scenarios.

In this chapter, we responded to research questions 4.1 to 4.3 as follows:

- *RQ 4.1: What methodology can be developed to estimate the quality of local user datasets, allowing the reporting of summarized information to the central server in federated learning, without compromising the privacy of raw data, in order to provide the server with a general understanding of the potential of local users' datasets?*

  We proposed a quality-estimator method that selectively involves a small group of users who possess high-quality data and robust training capacity in FL training rounds. Our proposed quality estimator metric analyzes mobility data in both the time and frequency domains. This metric generates a numerical value indicating the reliability of a user's data. Our solution offers a significant advantage as each local user can compute this metric on their data and only transmit the metric value to the central server, eliminating the need to transmit the actual data to increase privacy.

- *RQ 4.2: How can we optimize computational resource usage and minimize communication overhead in federated learning by devising a selection mechanism that chooses a subset of eligible local users to participate in each federated training round, while still maintaining representative and diverse dataset contributions?*

- *RQ 4.3: How can we design an efficient multi-objective neural architecture search algorithm for federated learning, aiming to develop high-performance models that concurrently optimize accuracy, training time, and transmission time, considering the varying network throughput and diverse datasets of federated participants, in order to achieve an optimal trade-off among these factors?*

  We proposed a transformer-based distributed trajectory predictor, called FedForce, which uses reinforcement learning to design the predictor's neural architecture based on the unique

mobility features, the computational resource availability, and the wireless network throughput. In this direction, we manipulate the reward function of the RL algorithm to incorporate a multi-objective cost function containing accuracy, training time, transmission time, and model size. This approach enables us to optimize such crucial factors simultaneously. We evaluated FedForce on two different datasets and showed that FedForce outperforms LSTM- and CNN-based trajectory predictors by 10% better accuracy and up to 70% lower training time. FedForce also outperforms non-NN models (e.g., RF, J48) by up to 20% better accuracy. We further show that FedForce converges to the same accuracy as the centralized training model, while the classical FL is up to 10% less accurate and takes 50% more training and transmission time. Moreover, FedForce achieves an ADE of 0.20m, which is lower than the best-performing TP baseline. FedForce also can save up to 80% of computational resources and 96% of communication overhead without remarkably reducing the mean accuracy.

For more comprehensive insights and detailed information, we recommend reading our FedForce paper [18], which extensively examines the subject matter.

### 10.2.5 Non-Cooperative Multi-Agent Trajectory Prediction

In Chapter 9 we introduce GTP-Force, a novel system for constructing a highly accurate trajectory predictor that captures the social interactions among multiple non-cooperative mobile users.

In this section, we responded to the research question 5 as follows:

- *RQ 5: How can we create a robust multi-agent trajectory prediction system empowered by neural architecture search, capable of accurately and efficiently capturing the dynamics of both cooperative and non-cooperative user behaviors?*

  We proposed a transformer-based non-cooperative trajectory predictor, called GTP-Force, which uses reinforcement learning to design the high-performance Transformer architecture based on the *intra-cluster* user mobility features. Subsequently, GTP-Force applies a non-cooperative game theory to determine the optimal combination of *inter-cluster* neural network architectures in the dynamic multi-agent environment. Our evaluation of GTP-Force is conducted on both small and large-scale scenarios, utilizing the ETH+UCY and Orange datasets, respectively. In the small-scale scenario, the proposed GTP-Force attains an ADE of 0.19, which corresponds to a lower positioning error when compared to several state-of-the-art models. In a large-scale scenario, our study demonstrates that Reinforced Transformers outperform both of the best baselines, LSTM- and CNN-based predictors, by achieving up to 10% higher accuracy and reducing training time by up to 70%. Furthermore, our proposed approach exceeds the performance of non-neural models based on XGBoost, RF, and J48 by up to 20% in terms of accuracy. Our trajectory predictor, based on the Reinforced Transformer in game theory, achieves 80% average accuracy, 5% higher than individual RL-TF, 10% higher than GT-TF without RL personalization, and 15% higher than cooperative social-TF without RL personalization. Our experiments show that by increasing the number of users in the system and subsequently in each cluster of similar behavior mobile users, GT-Force significantly reduces the number of training parameters compared to the case where

multiple RL-TFs are individually trained. This result suggests that our proposed approach is highly scalable and efficient, making it suitable for large-scale scenarios while maintaining high prediction accuracy.

For more comprehensive insights and detailed information, we recommend reading our GTP-Force paper [19], which extensively examines the subject matter.

## 10.3 Future Work

The potential applications of trajectory prediction are vast, and future research can explore new directions and innovative approaches for enhancing the accuracy and efficiency of these models. By addressing the challenges and opportunities presented by trajectory prediction, we can improve our understanding of human mobility patterns and develop more efficient and effective systems for intelligent transportation, modern wireless networks, and other critical areas. While our work has significantly improved the accuracy and efficiency of trajectory prediction models in various aspects, including personalized, multi-agent, distributed, Privacy-preserving, and non-cooperative scenarios, there is still significant room for improvement and further research. This section will outline potential future directions for trajectory prediction research, building upon our work and addressing remaining challenges in this field as follows.

- Isolated-Agent Trajectory Prediction:

- Context-Aware Prediction: It is required to enhance trajectory prediction models by incorporating contextual information, such as weather conditions, road conditions, and traffic patterns. By considering these factors, individuals can anticipate changes in their surroundings and adapt their trajectories accordingly.

- Multi-Agent Social-Aware Trajectory Prediction:

- Hybrid Models: It is required to explore the combination of rule-based models and data-driven approaches to enhance multi-agent trajectory prediction. Rule-based models can capture common social behaviors and conventions, while data-driven approaches can adapt to specific contexts and individual agent characteristics.
- Real-Time Adaptation: It is required to enhance multi-agent trajectory prediction models with real-time adaptation capabilities. Agents should be able to quickly update their predictions based on observed changes in the environment or the behaviors of other agents, ensuring dynamic and responsive navigation in complex social settings.

- Multi-Agent Federated Trajectory Prediction:

- Resource-Constrained Environments: It is required to optimize federated learning algorithms for resource-constrained environments, such as edge devices or autonomous vehicles with limited computational capabilities. For instance, efficient model compression and parameter quantization will be crucial for effective distributed trajectory prediction.

- Robustness and Security: It is required to address challenges related to robustness and security in federated learning-based trajectory prediction. Design mechanisms to detect and mitigate adversarial attacks, ensure data integrity and prevent model poisoning, maintaining reliability and trustworthiness.

- Multi-Agent Non-Cooperative Trajectory Prediction:

- Incentive Design: It is required to investigate incentive mechanisms and design strategies to encourage cooperative behavior among non-cooperative agents. By aligning individual interests with collective goals, trajectory prediction systems can promote more cooperative and socially beneficial behaviors, leading to improved overall traffic flow and efficiency.

# Bibliography


[1] 3GPP. "Evolved universal terrestrial radio access (e-utra); radio resource835control (rrc); protocol specification, 3GPP TS 36.331 V9.4.0 (2010-09)," in: *Evolved Universal Terrestrial Radio Access (E-UTRA);Radio Resource837Control (RRC);Protocol specification(Release 9)* (2011).

[2] Nadine Akkari and Nikos Dimitriou. "Mobility management solutions for 5G networks: Architecture and services". In: *Computer Networks* 169 (2020), p. 107082.

[3] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. "Social LSTM: Human Trajectory Prediction in Crowded Spaces". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[4] Javad Amirian, Jean-Bernard Hayet, and Julien Pettré. "Social ways: Learning multi-modal distributions of pedestrian trajectories with gans". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 0–0.

[5] Mohammad Bahram, Andreas Lawitzky, Jasper Friedrichs, Michael Aeberhard, and Dirk Wollherr. "A game-theoretic approach to replanning-aware interactive scene prediction and planning". In: *IEEE Transactions on Vehicular Technology* 65.6 (2015), pp. 3981–3992.

[6] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. "Designing neural network architectures using reinforcement learning". In: *arXiv preprint arXiv:1611.02167* (2016).

[7] Stuart J Barnes. "Location-based services: the state of the art". In: *E-Service* 2.3 (2003), pp. 59–70.

[8] C. Barrios, Y. Motai, and D. Huston. "Trajectory Estimations Using Smartphones". In: *IEEE Transactions on Industrial Electronics* 62.12 (Dec. 2015), pp. 7901–7910. ISSN: 0278-0046. DOI: 10.1109/TIE.2015.2478415.

[9] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[10] Guangshuo Chen, Aline Carneiro Viana, Marco Fiore, and Carlos Sarraute. "Complete trajectory reconstruction from sparse mobile phone data". In: *EPJ Data Science* 8.1 (2019), p. 30.

[11] Yan-Ying Chen, An-Jung Cheng, and Winston H Hsu. "Travel recommendation by mining people attributes and travel group types from community-contributed photos". In: *IEEE Transactions on Multimedia* 15.6 (2013), pp. 1283–1295.

[12] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. "Client selection in federated learning: Convergence analysis and power-of-choice selection strategies". In: *arXiv preprint arXiv:2010.01243* (2020).

[13] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555.

[14] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. "Multimodal trajectory predictions for autonomous driving using deep convolutional networks". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2090–2096.

[15] Thomas D'Roza and George Bilchev. "An overview of location-based services". In: *BT Technology Journal* 21.1 (2003), pp. 20–27.

[16] Nachiket Deo, Akshay Rangesh, and Mohan M. Trivedi. "How would surround vehicles move? A Unified Framework for Maneuver Classification and Motion Prediction". In: *CoRR* abs/1801.06523 (2018). arXiv: 1801.06523.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.

[18] Negar Emami, Antonio Di Maio, and Torsten Braun. "FedForce: Network-adaptive Federated Learning for Reinforced Mobility Prediction". In: *Accepted in International conference on Local Computer Networks (LCN 2023)*. IEEE. 2023.

[19] Negar Emami, Antonio Di Maio, and Torsten Braun. "GTP-Force: Game-Theoretic Trajectory Prediction through Distributed Reinforcement Learning". In: *Accepted in International conference on Mobile Ad-Hoc and Smart Systems (MASS 2023)*. IEEE. 2023.

[20] Negar Emami, Antonio Di Maio, and Torsten Braun. "INTRAFORCE: Intra-Cluster Reinforced Social Transformer for Trajectory Prediction". In: *2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE. 2022, pp. 333–338.

[21] Negar Emami, Lucas Pacheco, Antonio Di Maio, and Torsten Braun. "RC-TL: Reinforcement Convolutional Transfer Learning for Large-scale Trajectory Prediction". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2022, pp. 1–9.

[22] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. "Personalized federated learning: A meta-learning approach". In: *arXiv preprint arXiv:2002.07948* (2020).

[23] Jie Feng, Can Rong, Funing Sun, Diansheng Guo, and Yong Li. "PMF: A privacy-preserving human mobility prediction framework via federated learning". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.1 (2020), pp. 1–21.

[24] Stefano Ferretti, Vittorio Ghini, and Fabio Panzieri. "A survey on handover management in mobility architectures". In: *Computer Networks* 94 (2016), pp. 390–413.

[25] Ray J Frank, Neil Davey, and Stephen P Hunt. "Time series prediction and neural networks". In: *Journal of intelligent and robotic systems* 31 (2001), pp. 91–103.

[26] Zhipeng Gao, Jie Meng, Qian Wang, and Yang Yang. "Service migration for deadline-varying user-generated data in mobile edge-clouds". In: *IEEE World Congress on Services, SERVICES 2018* (2018), pp. 53–54. DOI: 10.1109/SERVICES.2018.00039.

[27] Philipp Geiger and Christoph-Nikolas Straehle. "Learning game-theoretic models of multiagent trajectories using implicit layers". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6. 2021, pp. 4950–4958.

[28] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. "Transformer networks for trajectory forecasting". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 10335–10342.

[29] Fanyu Gong, Ziwei Sun, Xiaodong Xu, Zhao Sun, and Xiaosheng Tang. "Cross-tier handover decision optimization with stochastic based analytical results for 5G heterogeneous ultra-dense networks". In: *2018 IEEE International Conference on Communications Workshops*. IEEE. 2018, pp. 1–6.

[30] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. "Understanding individual human mobility patterns". In: *nature* 453.7196 (2008), pp. 779–782.

[31] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. "Social gan: Socially acceptable trajectories with generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2255–2264.

[32] Emre Gures, Ibraheem Shayea, Abdulraqeb Alhammadi, Mustafa Ergen, and Hafizal Mohamad. "A comprehensive survey on mobility management in 5G heterogeneous networks: Architectures, challenges and solutions". In: *IEEE Access* 8 (2020), pp. 195883–195913.

[33] Dirk Helbing, Lubos Buzna, Anders Johansson, and Torsten Werner. "Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions". In: *Transportation science* 39.1 (2005), pp. 1–24.

[34] Dirk Helbing and Peter Molnar. "Social force model for pedestrian dynamics". In: *Physical review E* 51.5 (1995), p. 4282.

[35] Daniel S Hirschberg. "Algorithms for the longest common subsequence problem". In: *Journal of the ACM (JACM)* 24.4 (1977), pp. 664–675.

[36] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[37] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao. "Vehicle trajectory prediction based on motion model and maneuver recognition". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 4363–4369. DOI: 10.1109/IROS.2013.6696982.

[38] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. "Mobile edge computing—A key technology towards 5G". In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[39] Haifeng Jing, Yafei Zhang, Jiehan Zhou, Weishan Zhang, Xin Liu, Guizhi Min, and Zhanmin Zhang. "Lstm-based service migration for pervasive cloud computing". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1835–1840.

[40] Hadi S Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. "Hyp-rl: Hyperparameter optimization by reinforcement learning". In: *arXiv preprint arXiv:1906.11527* (2019).

[41] Mostafa Karimzadeh, Ryan Aebi, Allan M. de Souza, Zhongliang Zhao, Torsten Braun, Susana Sargento, and Leandro Villas. "Reinforcement Learning-designed LSTM for Trajectory and Traffic Flow Prediction". In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. 2021, pp. 1–6. DOI: 10.1109/WCNC49053.2021.9417511.

[42] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. "Social-Bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks". In: *Advances in Neural Information Processing Systems* 32 (2019).

[43] Matt Kusner, Jacob Gardner, Roman Garnett, and Kilian Weinberger. "Differentially private Bayesian optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 918–927.

[44] Billy Pik Lik Lau, Sumudu Hasala Marakkalage, Yuren Zhou, Naveed Ul Hassan, Chau Yuen, Meng Zhang, and U-Xuan Tan. "A survey of data fusion in smart city applications". In: *Information Fusion* 52 (2019), pp. 357–374.

[45] Stephanie Lefevre, Dizan Vasquez, and Christian Laugier. "A survey on motion prediction and risk assessment for intelligent vehicles". In: *Robomech Journal* 1.1 (), pp. 1–14.

[46] Florin Leon and Marius Gavrilescu. "A review of tracking and trajectory prediction methods for autonomous driving". In: *Mathematics* 9.6 (2021), p. 660.

[47] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. "Crowds by Example". In: *Computer Graphics Forum* 26.3 (2007), pp. 655–664. DOI: https://doi.org/10.1111/j.1467-8659.2007.01089.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01089.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01089.x.

[48] Chunlin Li, Lei Zhu, Weigang Li, and Youlong Luo. "Joint edge caching and dynamic service migration in SDN based mobile edge computing". In: *Journal of Network and Computer Applications* 177.July 2020 (2021), p. 102966. ISSN: 10958592. DOI: 10.1016/j.jnca.2020.102966. URL: https://doi.org/10.1016/j.jnca.2020.102966.

[49] Zezu Liang, Yuan Liu, Tat Ming Lok, and Kaibin Huang. "Multi-Cell Mobile Edge Computing: Joint Service Migration and Resource Allocation". In: *IEEE Transactions on Wireless Communications* 20.9 (2021), pp. 5898–5912. ISSN: 15582248. DOI: 10.1109/TWC.2021.3070974. arXiv: 2102.03036.

[50] Massimiliano Luca, Gianni Barlacchi, Bruno Lepri, and Luca Pappalardo. "A survey on deep learning for human mobility". In: *ACM Computing Surveys (CSUR)* 55.1 (2021), pp. 1–44.

[51] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[52] 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". In: *IEEE Transactions on Vehicular Technology* 69.2 (2020), pp. 2126–2140. DOI: 10.1109/TVT.2019.2960313.

[53] Wei-Chiu Ma, De-An Huang, Namhoon Lee, and Kris M Kitani. "Forecasting interactive dynamics of pedestrians with fictitious play". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 774–782.

[54] Nathalie Majcherczyk, Nishan Srishankar, and Carlo Pinciroli. "Flow-fl: Data-driven federated learning for spatio-temporal predictions in multi-robot systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8836–8842.

[55] Mahmoud Mandour, Fayez Gebali, Ashraf D Elbayoumy, Gamal M Abdel Hamid, and Amr Abdelaziz. "Handover Optimization and User Mobility Prediction in LTE Femtocells Network". In: *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2019, pp. 1–6.

[56] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. "It is not the journey but the destination: Endpoint conditioned trajectory prediction". In: *European Conference on Computer Vision*. Springer. 2020, pp. 759–776.

[57] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[58] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5 (2001), pp. 64–67.

[59] Abdelrahim Mohamed, Oluwakayode Onireti, Seyed Amir Hoseinitabatabaei, Muhammad Imran, Ali Imran, and Rahim Tafazolli. "Mobility prediction for handover management in cellular networks with control/data separation". In: *2015 IEEE International Conference on Communications (ICC)*. IEEE. 2015, pp. 3939–3944.

[60] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. "Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 14424–14432.

[61] Brendan Tran Morris and Mohan Manubhai Trivedi. "A survey of vision-based trajectory learning and analysis for surveillance". In: *IEEE transactions on circuits and systems for video technology* 18.8 (2008), pp. 1114–1127.

[62] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Haifa, Israel, 2010, pp. 807–814. ISBN: 978-1-60558-907-7.

[63] Van-Linh Nguyen, Po-Ching Lin, and Ren-Hung Hwang. "Enhancing misbehavior detection in 5G vehicle-to-vehicle communications". In: *IEEE Transactions on Vehicular Technology* 69.9 (2020), pp. 9417–9430.

[64] Nishant Nikhil and Brendan Tran Morris. "Convolutional neural network for trajectory prediction". In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0.

[65] OpenAI. "ChatGPT Language Model". In: *ChatGPT-3*. 2023. URL: https://openai.com/chatgpt.

[66] Tao Ouyang, Zhi Zhou, and Xu Chen. "Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing". In: *IEEE Journal on Selected Areas in Communications* 36.10 (2018), pp. 2333–2345. ISSN: 15580008. DOI: 10.1109/JSAC.2018.2869954. arXiv: 1809.05239.

[67] Tao Ouyang, Zhi Zhou, and Xu Chen. "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing". In: *IEEE Journal on Selected Areas in Communications* 36.10 (2018), pp. 2333–2345.

[68] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. "You'll never walk alone: Modeling social behavior for multi-target tracking". In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 261–268.

[69] D. J. Phillips, T. A. Wheeler, and M. J. Kochenderfer. "Generalizable intention prediction of human drivers at intersections". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. June 2017, pp. 1665–1670. DOI: 10.1109/IVS.2017.7995948.

[70] Derek J Phillips, Tim A Wheeler, and Mykel J Kochenderfer. "Generalizable intention prediction of human drivers at intersections". In: *2017 IEEE intelligent vehicles symposium (IV)*. IEEE. 2017, pp. 1665–1670.

[71] Carole G Prevost, Andre Desbiens, and Eric Gagnon. "Extended Kalman filter for state estimation and trajectory prediction of a moving object detected by an unmanned aerial vehicle". In: *2007 American control conference*. IEEE. 2007, pp. 1805–1810.

[72] Shaojie Qiao, Dayong Shen, Xiaoteng Wang, Nan Han, and William Zhu. "A self-adaptive parameter selection trajectory prediction approach via hidden Markov models". In: *IEEE Transactions on Intelligent Transportation Systems* 16.1 (2014), pp. 284–296.

[73] Andrey Rudenko, Luigi Palmieri, Michael Herman, Kris M Kitani, Dariu M Gavrila, and Kai O Arras. "Human motion trajectory prediction: A survey". In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935.

[74] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. "Sophie: An attentive gan for predicting paths compliant to social and physical constraints". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1349–1358.

[75] Muhammad Saleem, Sagheer Abbas, Taher M Ghazal, Muhammad Adnan Khan, Nizar Sahawneh, and Munir Ahmad. "Smart cities: Fusion-based intelligent traffic congestion control system for vehicular networks using machine learning techniques". In: *Egyptian Informatics Journal* 23.3 (2022), pp. 417–426.

[76] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel, and K. Kuhnert. "When will it change the lane? A probabilistic regression approach for rarely occurring events". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. June 2015, pp. 1373–1379. DOI: 10.1109/IVS.2015.7225907.

[77] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[78] Ibraheem Shayea, Mustafa Ergen, Marwan Hadri Azmi, Sultan Aldirmaz Çolak, Rosdiadee Nordin, and Yousef Ibrahim Daradkeh. "Key challenges, drivers and solutions for mobility management in 5G networks: A survey". In: *IEEE Access* 8 (2020), pp. 172534–172552.

[79] Aditya Shrivastava, Jai Prakash V Verma, Swati Jain, and Sanjay Garg. "A deep learning based approach for trajectory estimation using geographically clustered data". In: *SN Applied Sciences* 3.6 (2021), pp. 1–17.

[80] Xiang Sun and Nirwan Ansari. "EdgeIoT: Mobile edge computing for the Internet of Things". In: *IEEE Communications Magazine* 54.12 (2016), pp. 22–29.

[81] Cynthia Sung, Dan Feldman, and Daniela Rus. "Trajectory clustering for motion prediction". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1547–1552.

[82] Canh T Dinh, Nguyen Tran, and Josh Nguyen. "Personalized federated learning with moreau envelopes". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21394–21405.

[83] Luca Tartarini, Marcelo Antonio Marotta, Eduardo Cerqueira, Juergen Rochol, Cristiano Bonato Both, Mario Gerla, and Paolo Bellavista. "Software-defined handover decision engine for heterogeneous cloud radio access networks". In: *Computer Communications* 115 (2018), pp. 21–34.

[84] Ranjeet Singh Tomar, Shekhar Verma, and Geetam Singh Tomar. "SVM based trajectory predictions of lane changing vehicles". In: *2011 International Conference on Computational Intelligence and Communication Networks*. IEEE. 2011, pp. 716–721.

[85] Javad Akbari Torkestani. "Mobility prediction in mobile wireless networks". In: *Journal of Network and Computer Applications* 35.5 (2012), pp. 1633–1645.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[87] Chunnan Wang, Xiang Chen, Junzhe Wang, and Hongzhi Wang. "ATPFL: Automatic Trajectory Prediction Model Design Under Federated Learning Framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 6563–6572.

[88] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. "A survey on service migration in mobile edge computing". In: *IEEE Access* 6 (2018), pp. 23511–23528.

[89] Jürgen Wiest, Matthias Höffken, Ulrich Kreßel, and Klaus Dietmayer. "Probabilistic trajectory prediction with Gaussian mixture models". In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE. 2012, pp. 141–146.

[90] Bowen Xie, Yuxuan Sun, Sheng Zhou, Zhisheng Niu, Yang Xu, Jingran Chen, and Deniz Gündüz. "MOB-FL: Mobility-Aware Federated Learning for Intelligent Connected Vehicles". In: *arXiv preprint arXiv:2212.03519* (2022).

[91] 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". In: *IEEE Transactions on Industrial Electronics* 65.7 (July 2018), pp. 5999–6008. ISSN: 0278-0046. DOI: 10.1109/TIE.2017.2782236.

[92] Rui Xu and Donald Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678.

[93] Xiaodong Xu, Xiaoxuan Tang, Zhao Sun, Xiaofeng Tao, and Ping Zhang. "Delay-Oriented Cross-Tier Handover Optimization in Ultra-Dense Heterogeneous Networks". In: *IEEE Access* 7 (2019), pp. 21769–21776. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2898430.

[94] Cunjun Yu, Xiao Ma, Jiawei Ren, Haiyu Zhao, and Shuai Yi. "Spatio-temporal graph transformer networks for pedestrian trajectory prediction". In: *European Conference on Computer Vision*. Springer. 2020, pp. 507–523.

[95] Xiang Yu, Maolin Guan, Mingxia Liao, and Xia Fan. "Pre-Migration of Vehicle to Network Services Based on Priority in Mobile Edge Computing". In: *IEEE Access* 7 (2019), pp. 3722–3730. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2888478.

[96] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[97] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. "A review of moving object trajectory clustering algorithms". In: *Artificial Intelligence Review* 47.1 (2017), pp. 123–144.

[98] Simone Zamboni, Zekarias Tilahun Kefato, Sarunas Girdzijauskas, Noren Christoffer, and Laura Dal Col. "Pedestrian Trajectory Prediction with Convolutional Neural Networks". In: *arXiv preprint arXiv:2010.05796* (2020).

[99] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. "Deep learning in mobile and wireless networking: A survey". In: *IEEE Communications surveys & tutorials* 21.3 (2019), pp. 2224–2287.

[100] Weishan Zhang, Pengcheng Duan, Laurence T. Yang, Feng Xia, Zhongwei Li, Qinghua Lu, Wenjuan Gong, and Su Yang. "Resource requests prediction in the cloud computing environment with a deep belief network". In: *Software - Practice and Experience* 47.3 (2017), pp. 473–488. ISSN: 1097024X. DOI: 10.1002/spe.2426.

[101] Weishan Zhang, Shouchao Tan, Qinghua Lu, Xin Liu, and Wenjuan Gong. "A genetic-algorithm-based approach for task migration in pervasive clouds". In: *International Journal of Distributed Sensor Networks* 11.8 (2015), p. 463230.

[102] Weishan Zhang, Shouchao Tan, Feng Xia, Xiufeng Chen, Zhongwei Li, Qinghua Lu, and Su Yang. "A survey on decision making for task migration in mobile cloud environments". In: *Personal and Ubiquitous Computing* 20.3 (2016), pp. 295–309.

[103] Wenjing Zhang, Yuan Liu, Tingting Liu, and Chenyang Yang. "Trajectory prediction with recurrent neural networks for predictive resource allocation". In: *2018 14th IEEE International Conference on Signal Processing (ICSP)*. IEEE. 2018, pp. 634–639.

[104] Zhongliang Zhao, Negar Emami, Hugo Santos, Lucas Pacheco, Mostafa Karimzadeh, Torsten Braun, Arnaud Braud, Benoit Radier, and Philippe Tamagnan. "Reinforced-LSTM Trajectory Prediction-driven Dynamic Service Migration: A Case Study". In: *IEEE Transactions on Network Science and Engineering* (2022).

[105] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).

# List of publications


- Zhao, Z., N. Emami, H. Santos, L. Pacheco, M. Karimzadeh, T. Braun, A. Braud, B. Radier, and P. Tamagnan (2022). Reinforced-lstm trajectory prediction-driven dynamic service migration: A case study. IEEE Transactions on Network Science and Engineering 9 (4), 2786–2802.
  DOI: 10.1109/TNSE.2022.3169786.

- Emami, N., L. Pacheco, A. Di Maio, and T. Braun (2022). Rc-tl: Reinforcement convolutional transfer learning for large-scale trajectory prediction. In NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, pp. 1–9. IEEE.
  DOI: 10.1109/NOMS54207.2022.9789883 .

- Emami, N., A. Di Maio, and T. Braun (2022). Intraforce: Intra-cluster reinforced social transformer for trajectory prediction. In 2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 333–338. IEEE.
  DOI: 10.1109/WiMob55322.2022.9941547.

- Emami, N., A. Di Maio, and T. Braun (2023). Fedforce: Network-adaptive federated learning for reinforced mobility prediction. Accepted in International conference on Local Computer Networks (LCN 2023). IEEE.
  DOI: 10.48350/179855.

- Emami, N., A. Di Maio, and T. Braun (2023). GTP-Force: Game-Theoretic Trajectory Prediction through Distributed Reinforcement Learning. Accepted in International Conference on Mobile Ad-Hoc and Smart Systems (MASS 2023). IEEE.
  DOI: 10.48350/182255.

# Declaration of consent

on the basis of Article 18 of the PromR Phil.-nat. 19

Name/First Name: Emami Negar

Registration Number: 19-124-122

Study program: Computer Science

Bachelor ☐ Master ☐ Dissertation ☑

Title of the thesis: Deep Learning Techniques for Mobility Prediction and Management in Mobile Networks

Supervisor: Prof. Dr. Torsten Braun

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of September 5th, 1996 and Article 69 of the University Statute of June 7th, 2011 is authorized to revoke the doctoral degree awarded on the basis of this thesis.
For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with theses submitted by others.

Bern, 25.07.2023

Place/Date

Negar Emami Digitally signed by Negar Emami
Date: 2023.07.25 16:28:12 +02'00'

Signature