SEMANTIC

end-to-end Slicing and data-drivEn autoMAtion of Next generation cellular neTworks with mobIle edge Clouds

Marie Skłodowska-Curie Actions (MSCA)
Innovative Training Networks (ITN)
H2020-MSCA-ITN-2019
861165 - SEMANTIC

WP4 – Data-driven network control and automation

D4.3: Performance evaluation of automated network control and proof-of-concept

Contractual Date of Delivery:	M49
Actual Date of Delivery:	25/12/2023
Responsible Beneficiary:	EURECOM
Contributing beneficiaries:	EUR, TLN, IQU, NOK
Security:	Confidential
Nature:	Report
Version:	V1.0

Document Information

Version Date: 25/12/2023
Total Number of Pages: 53

Authors

Name	Organization	Email
Ali Ehsanian	EURECOM	ali.ehsanian@eurecom.fr
Maryam Bandali	TELENOR	maryam.bandali@telenor.se
Sudharshan Paindi Jayakumar	NOKIA	sudharshan.paindi_jayakumar@nokia.com
Navideh Ghafouri	IQUADRAT	n.ghafoori@iquadrat.com

Document History

Revision	Date	Modification	Contact Person
V0.1	18/12/2023	Individual contributions from ESRs	Ali Ehsanian
V0.2	20/12/2023	First draft for review	Ali Ehsanian
V1.0	25/12/2023	Released version	Ali Ehsanian

Table of Contents

List of Figures

List of Tables

Executive summary

Deliverable D4.3 includes the SEMANTIC ESRs contributions to WP4 (Data-driven network control and automation). More specifically, first deliverable (D4.1) described the state of the art pertinent to WP4 scope, and second deliverable (D4.2) explained the specific research questions on which the ESR focused, the approaches they adopted, and their proposed solutions. The third deliverable (D4.3) presents the simulations and analytical results of the proposed solutions. Depending on the maturity of the work of each ESRs, some results are presented.

Section 2 presents the work currently being done at IQUADRAT by ESR Navideh Ghafouri. After a review of the history of the research work in the previous deliverable, this section provides the implementation technicalities of the proposed scheme. The simulation details and numerical results are presented, and the section continues by conclusion and future steps. f

In section 3 Sudharshan Paindi Jayakumar (ESR in Nokia) proposes novel methodologies for optimizing and managing base station parameters in mobile communication networks. CEDA-BatOp, a framework for base station parameter optimization and automation, employs a machine learning approach to optimize base station parameters and enhance network performance. CEDA-BatOp 2.0, an extension of CEDA-BatOp, incorporates joint optimization, controlled drift analysis, and pseudo-labeling techniques to further improve network performance and address potential issues arising from parameter drift. Additionally, the work proposes an anomaly detection system based on Deep Generalized Canonical Correlation Analysis (DGCCA) to effectively identify anomalies in UE parameters, promoting network reliability and resilience. The proposed methodologies evaluated using simulated and real-world datasets, demonstrating their efficacy in optimizing base station parameters, mitigating parameter drift, and detecting anomalies in UE parameters. The results showcase significant improvements in network performance metrics, including throughput, latency, and energy efficiency. These findings highlight the potential of machine learning and advanced anomaly detection techniques for optimizing and managing base station parameters, paving the way for more efficient and reliable mobile communication networks.

Section 4 describes the proof of concept for the methodology and evaluation carried on at Telenor by ESR Maryam Bandari. The ESR implement a Machine Learning (ML) based approach to help with network automation and management in the network disturbance domain. Many network disturbances are flooding the Network Operation Centers (NOC) of telecommunication (telco) companies in the form of network Trouble Tickets (TT). Automation plays an important role for managing these TTs to increase Quality of Service (QoS) and subsequently, Quality of Experience (QoE). Currently, there is a gap in literature for developing automated assistant systems for handling network TTs. In this work, we manage to develop a solution to address two challenges related to TTs generated from fixed and mobile access network domains: 1. prediction of resolution times and 2. prediction of onsite dispatch work needs. For the first use case, we come up with a 60-minute confidence interval and succeed to predict the correct resolution time ranges in 90% (fixed) and 80% (mobile) of the cases at TT creation time. Therefore, we have an 80% and 61% improvement over company baseline. In second use case, there is no company baseline, and we achieve an average macro F1 score of 65% and 70% respectively. This indicates a significant optimization in workforce (resource) usage of the company. We also study the evolution of access switch TTs over time and the insights that can be captured as more information is added. We realize due to update of the information within 15 minutes of TT creation, we improve the results by 57% and 50% respectively in first and second use cases.

Finally, the work done by ESR Ali Ehsanian at Eurecom is described in section 5, where the problem of resource allocation when several network slices compete for shared network infrastructure is addressed. Network slicing enables efficient and dynamic service deployment and network management of 5G and beyond 5G networks. It requires the capability to allocate to each slice the desired resources. This task is in reality very complicated, where the traditional approaches struggle to properly manage resource allocation because of the lack of precise models and hidden problem structures. Section 5 presents a data driven

algorithm for optimal resource allocation between slices in sliced virtualized mobile networks, to avoid under-provisioning and overprovisioning. The approach is based on a Distributed Deep Neural Network (DDNN) architecture which is distributed over edge and the cloud that attempts to exploit correlations between the demands of different slices/resources. The proposed DDNN is a DNN with multiple exit points; one local exit (e.g. in RAN) and one remote exit (e.g. in MEC). The DDNN needs to be trained jointly in order to achieve the desired goals. In the joint training a weight is assigned to local exit and a weight to remote exit, which forces the local exit to have good performance and also affects the remote exit performance. Also, the objective function plays an important role to avoid underprovisioning. The work introduce two different confidence mechanisms that in the online mode can decide just based on the output of the local exit (local predictions) either the local exit output is good enough to be used for resource allocation or the data must be sent to the remote layers where a better decision can be made. Performances evaluated through simulation and compared with state of art.

1 Introduction and document structure

SEMANTIC “end-to-end Slicing and data-drivEn autoMAtion of Next generation cellular neTworks with mobIle edge Clouds” is a H2020 ITN project funded by the EU, which aims to create an innovative research and training network for multi-GHz spectrum communications, MEC-empowered service provisioning and end-to-end network slicing, all integrated and jointly orchestrated by forward-looking data-driven network control and automation exploiting the enormous amounts of mobile big data spurred into the mobile data network.

In this document, SEMANTIC ESRs contribute to the deliverable D4.3 titled “Performance evaluation of automated network control and proof-of-concept” towards the objectives of WP4 (Data-driven network control and automation). This document summarizes the developments in key findings of the ESRs towards the task 4.3. This includes designing and implementing data-driven algorithmic framework to combine machine learning with model-base optimizations for predicting network performance, detecting anomalies and to prevent failures.

2 Multi-level Network Slicing and Resource Management in 6G

2.1 Introduction

This section starts with a review of the last deliverable. As mentioned in deliverable 4.2, we consider an O-RAN-based architecture for the 6G system model, and we proposed a network slicing and resource management benefiting from the programmability and openness of ORAN. The general approach consists of two levels: centralized decision-making and decentralized slice realization. The preliminaries, and enabling technologies were presented in the previous deliverable. To continue, Figure 1 shows the system model and where each level of management is located. In the next section of this deliverable, we present the implementation technicalities of the general idea. Simulation details and numerical results are provided and future steps are presented in the end.

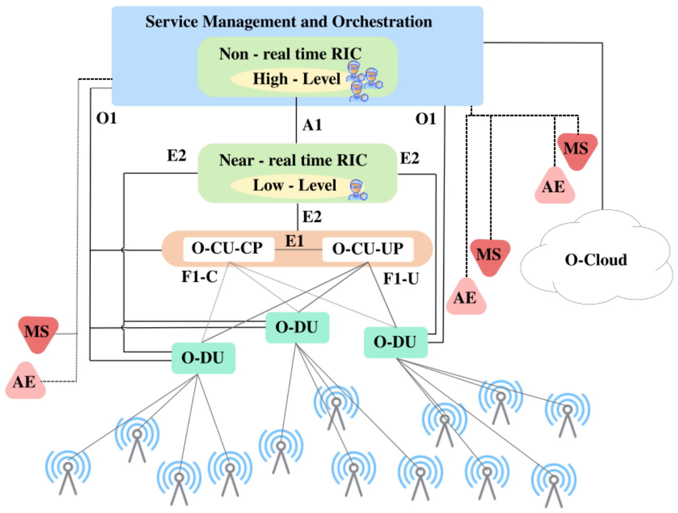


Figure 1 - System model and agent placements

2.2 The Implementation Technicalities and Numerical Results

Following the description of how the proposed scheme fits into the system model in deliverable 4.2, the technical details of the implementation need to be taken care of.

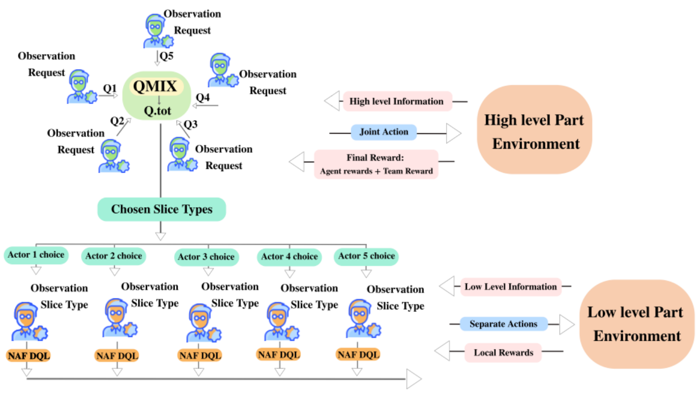


Figure 2 - The proposed approach: Agents and environments communication.

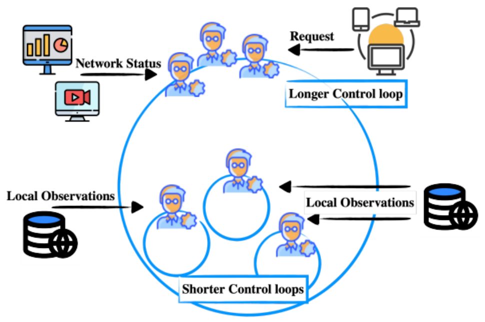


Figure 3 - The proposed approach: Time slots and control loops.

As shown in Figures 2 and 3, both levels are described as follows:

High-level Part: As mentioned earlier, 6G networks will consider additional service types to the three main types already available in 5G. Thus, according to the service types suggested in [1], we consider 5 slice types for our 6G environment, including FeMBB: further-enhanced mobile broadband, umMTC: ultra-massive machine-type communications, ERLLC: extremely reliable and low-latency communications, LDHMC: long-distance and high-mobility communications, and ELPC: extremely low-power communications; corresponding use-cases and KPIs for each service type are listed in Table 1.

Table 1 - Envisioned service types for 6G

1	FeMBB	Holographic Verticals, Full-Sensory Digital, Reality (VR/AR), Tactile/Haptic Internet, UHD/EHD Videos	Peak Data Rate: > 1 Tb/s User-Experienced Data rate: 1Gb/s Area Traffic Capacity: 1Gps/m^2 SE: 5-10 A
2	umMTC	IoE, Smart City/Home	Latency: 10-100 us Connectivity Density: 107 d/km^2 EE: 10-100 A
3	ERLLC	Fully Automated Driving, Industrial Internet	Latency: 10-100 us Mobility: > 1000 km/h Connectivity Density: 107 d/km^2
4	LDHMC	Deep-Sea Sightseeing, Space Travel, Hyper HSR	Mobility: > 1000 km/h

5	ELPC	Nanodevices, Nanorobots, Nanosensors, e-Health	Connectivity Density: 107 d/km^2 EE: 10-100 A

The number of agents at the high-level part is relative to multiple criteria, including how many tenants at the same time should be supported, in addition to the number of service types and their popularity in the network. This is because each agent receives one request in every longer time slot, thus, the total received requests equals the number of agents. In the following, we consider 5 agents in the MARL set to estimate equal requests for each service type in every longer time slot. In this way, we predict approximately one request for each service type. However, this number can change in case of having more service types or more number of users. The observation of each agent includes a request in addition to available service types. The request can be just one or a combination of defined KPIs, as reported in Table I. For example, a request can correspond to a latency requirement of 10-100 µs, meaning the agent can assign one of umMTC or ERLLC. Following the goal of maximizing the use of network capacity, the agent will make an optimal choice that gives a better result in cooperation with other agents' requests and decisions. Each agent tries to find the proper slice type according to the request. This means that no assignment will happen if the resources for the requested slice type are unavailable in the system. As a result, if agents assign the proper slice type or wait in case of no availability, they will receive their individual rewards. However, the general joint goal is to maximize the use of available resources and thus maximize the network capacity. Consequently, the final reward of this level is a combination of each agent's reward and the reward the team receives based on how much resource have been assigned at the end of the long time slot. We consider the higher ratio of the final reward for the joint goal to encourage allocating more resources while guiding the agents to make decisions. Accordingly, The high-level reward is calculated as follows:

$$R_{tot-agent_i} = \frac{1}{3} * R_{agent_i} + \frac{2}{3} * R_{team} \qquad (1)$$

in which R_{agent_i} is equal to 1 if the assigned resources are available. Otherwise, the agent's individual reward is 0. R_{team}, on the other hand, is the ratio of resources in-use to all resources, which means more resources in-use result in a better reward. The considered ratio is the final decision of various trials with different ratios.

One of the main problems in multi-agent settings is the exponential growth of joint action spaces with the number of agents. To overcome this complexity, we consider the QMIX algorithm [2] for our MARL setting, which lies between fully centralized and independent MARL. This means the algorithm learns decentralized policies in a centralized fashion and represents complex centralized action-value functions in a factored manner. Moreover, it does not require on-policy learning and, thus, remains practical even when deploying more agents [3]. It is worth mentioning that fully cooperative MARL is an active area of research. Still, there is a challenge of providing centralized training for agents to find optimum global policy while ensuring decentralized execution. QMIX is similar to Value Decomposition Networks (VDNs) [4] since they both can learn a centralized but factored Q.total-value by representing the Q.total-value as the sum of individual value functions that condition only on individual observations and actions. Thus, VDN also lies between independent Q-Learning and centralized Q-Learning, but in QMIX, the full factorization of VDN is not needed to extract decentralized but fully consistent policies. QMIX is made of agents´networks producing each agents´ Q-value and a mixing network that combines them in a complex, non-linear way to ensure consistency between centralized and decentralized policies. It is worth emphasising that this MARL differs from FRL, which involves training a ML model across multiple decentralized edge devices. In this MARL set, multiple agents interact with a common environment simultaneously, so the existence of other agents affects the environment of each agent continuously. Also, the considered algorithm is cooperative but does not need a joint action space. Instead, each agent has a network producing a Q-value and a mixing network that receives the Q-values and produces the team Q-value. Later, the team Q-value can be presented with each agent's Q-value again. Thus, The chosen team Q-value produces the individual Q-values that produced that highest value. This will be mapped to the related slice type.

Low-level Part: While at the high-level part the multi-agent set tries to maximize the network capacity by choosing proper slice types to the requests, the agents in the low-level part are responsible for providing the QoS of each slice by assigning the resource blocks during that longer time step. To achieve this objective, every agent tries allocating available resource blocks in shorter time steps. In other words, by defining a proper action space low limit and high limit, the agent will apply the assignment so that all KPIs defining that slice are in the proper range. Each single agent receives its local reward since at this level agents act independently. The local reward starts from zero and the highest amount is when all predefined KPIs are in their proper range. Meaning, the local reward starts from zero, and increases with every KPI being in the desired range.

Considering the complexity of assigning resource blocks in this architecture, the best way to address this difficulty is to use learning agents inspired by the psychology of human learning. DQL, as a model-free off-policy DRL algorithm that uses both Experience Replay and Network Cloning, has shown good sample efficiency and stable performance [5]. Since the DQL agent can collect information and train its policy in the background, the learned policy stored in the neural network can be easily transferred to the situations [6].

The benefits of DQL aside, our high-level algorithm is a Q-based algorithm with replay memory. Since the execution level is in a nested structure, having the same nature and procedure at both levels decreases the possibility of inconsistencies and improves the harmony of the general performance. DQL uses a discrete action space, which is improper for most real-world problems. To have a compatible method with our continuous action space in the low-level part, we use DQL with the help of Normalized Advantage Function (NAF DQL) [7]. NAF DQL is DQL compatible with continuous action space environments. While in regular DQNs, the output demonstrates all possible actions, and later, the highest value is chosen, in NAF, the neural network estimates the value function and the Advantage. Combining two streams produces the Q-value, and then the argmax is taken. According to [7], NAF DQL outperforms DDPG in solving the majority of tasks. Having all the benefits of Q-learning and its superior performance in problems with continuous action space assures that NAF DQL is a compatible algorithm for the low-level part.

The selected algorithms in both levels benefit from the experience replay mechanism, which refers to the case where experiences are stored in replay memories. At the high-level part, Q.tot-values in the memory help agents select the best service types based on previous similar states and observations. At the low-level part, each experience in the memory helps agents to know the best sequence of assigning actions to realize that service type.

2.3 Simulation Details and Numerical Results

In this section, we evaluate the proposed solution through numerical simulations. These simulations use Python 3.9 with PyTorch and PyTorch Lightening libraries. Py-Charm IDE, the Anaconda platform, and Google Colab have been used for coding. Since each level of the proposed scheme has access to different representations of the network, two different environments have been developed with the help of OpenAI gym. The environment for the high-level part interacts with the QMIX algorithm, and the environment for the low-level part interacts with single agents using NAF DQL [2], [7]. This simulation does not implement all structures of the O-RAN. We consider O-RAN as an enabling technology for 6G networks to be able to use our RL-based approach in the RICs. instead, each environment simulates the data that algorithms receive from interacting with an envisioned 6G network. Having CF mMIMO makes the environment have RUs and DUs that can be assigned to multiple users as long as they do not share one resource block with multiple users.

It is worth mentioning that this simulation is one implementation of the general idea proposed in this paper. The environments and algorithms can be changed or improved.

In our simulation, in the high-level part, as mentioned in Section III, the state space consists of the general information of the network status, which are the service types as a group of KPIs mentioned in Table 1. At the same time, the observation of each agent includes a request. Without loss of generality, each request asks only for one KPI in our simulation. So sates and observations are represented as:

$$state\ space = \{Service\ types\} \qquad (2)$$

$$observation\ space = \{r_i, (Service\ types)\} \quad (3)$$

in which Service types are sets of KPIs and r_i is also one KPI.

Action is choosing one of the predefined service types, which is a discrete number between 0 and 4. The next state is all the KPIs affected by removing the chosen service type and its related resources. Since the reward system was discussed in Section III, we avoid it here and follow this section with the last remaining element of the 4-tuple Markov Decision Process $P_a(S, S)$, which is the probability that action a in state S at time t will lead to state S´ at time t+1. Similar to most reinforcement learning cases, it is challenging to represent the transition probability distributions; instead, an episodic simulator can be used. As a result, both levels of this simulation benefit from episodic environment simulators that can be started from an initial state and yield a subsequent state and reward every time they receive an action input.

There exist five agents in our simulation; each one´s network produces a Q-value regarding the slice type it chooses for the request received in its observation. According to the QMIX algorithm, the Q-values will be fed to the mixing network. The weights and biases of the mixing network are produced by hyper-networks, which use the state and generate the layers. Since the weights should be non-negative, the leaner layer is followed by an absolute activation function. The final bias is also followed by a ReLU non-linearity. The mixing network and each agent´s network have been created according to [2].

After running a set of random trials, we chose the set of values 0.00001, 0.90, and 0.70 for learning rate, gamma, and epsilon which gave us the best training result. In the training phase, every episode consists of 4 cycles, each ending whenever all the resources are in use, and no free resource is available. The stored experience at this level consists of current and next observations and states, in addition to actions and rewards. Figure 4 illustrates the loss and reward plots of the multi-agent set in a 7500-episode run. The loss plot converged after around 5000 episodes, though. The descending loss plot and ascending reward plot show that our ML algorithm works properly in the simulated environment. In addition, the reward plot's importance comes from the fact that this level's reward system presents both agent and team rewards. The general goal for the mixing network is to maximize the number of assignments, which means each assignment should be optimal so that more users are served. Thus, the ascending plot shows the assigned resources corresponding to the network capacity increase. As mentioned before, increasing the number of accepted assignments means increasing network capacity in this research work. It should be noted that in this simulation if the requests repeat asking for the same service type, which can be unavailable after a while, agents will receive their rewards for not assigning the wrong slice type. Still, the total reward may decrease since there are no free resources. This explains the multiple rises and falls of the reward in Figure 4. The red linear trend-line has an increasing pattern, though.

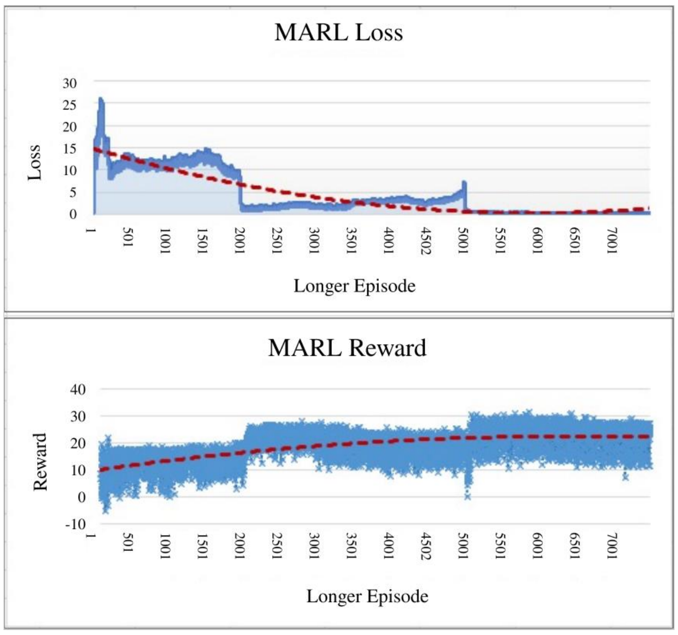


Figure 4 - Loss and Reward at the high-level part.

While the MARL algorithm at the high-level part selects the slice types in longer time steps, at the low-level part, the environment provides information related to resources that can be assigned. The state and observation spaces are the same at the beginning of each episode and present continuous values representing accessible resource blocks. Each agent has access to a limited number of resource blocks. The resources accessed by each agent are the ones located closer to the user. Contrary to the first level, action space is continuous and consists of eight (equal to the number of KPIs) positive and negative numbers. each array of actions changes the state values until each value is in the proper range for the service type. Similar to the first environment, at this level, the environment will produce the next state according to the input action and its effects on the available resources. Using the proper action space limits after multiple trials, and checking constraints in the reward system assign the optimum amount of resources to each request. In other words, since the high-level part monitors the availability and makes decisions, and the execution is decentralized by single agents at the low-level part, the agents will not just guarantee a lower limit but also an upper limit. Thus, as defined in the reward system of the low-level part environment, for an agent to receive its reward, the KPIs should be between a maximum and a minimum level.

In our simulation, all single agents are the same, but each one is trained to realized one of service types. In the training phase, each episode consists of a maximum of 4 steps. At the beginning of the training process, in the trade-off between exploration and exploitation, each agent mostly uses random actions to explore and sample more data. Thus, the value of epsilon starts from 1 and reduces over passing epochs until the agent mostly uses Q-values to take actions from replay memory. Each experience in the replay memory consists of the current and next observation, action, and reward. In order to do that, we reduce the initial random sample probability to over \$100\$ epochs. In other words, in the training-epoch-end method of PyTorch Lightening, epsilon will be chosen according to the:

$$\max\left\{epsilonMin, epsilonStart - \frac{currentEpoch}{100}\right\} \qquad (4)$$

Based on (4), the random sampling probability will start at its highest value and decrease during 100 epochs until it reaches the minimum value considered for epsilon. We used the Optuna library along with PyTorch Lightening to perform 20 trials to find the best value for learning rate (0.00015193) and gamma (0.011332).

Using the aforementioned values of learning rate and gamma, Figure 5 shows the Loss plots for all five single agents in 24000 epochs, each learning to realize one slice type. At the beginning of training, the plots for each service type increase due to taking random actions. However, as the training continues, after approximately 2000 epochs, the loss for each agent starts a descending pattern, and finally, after 6000 episodes, all plots converge to the minimum amount.

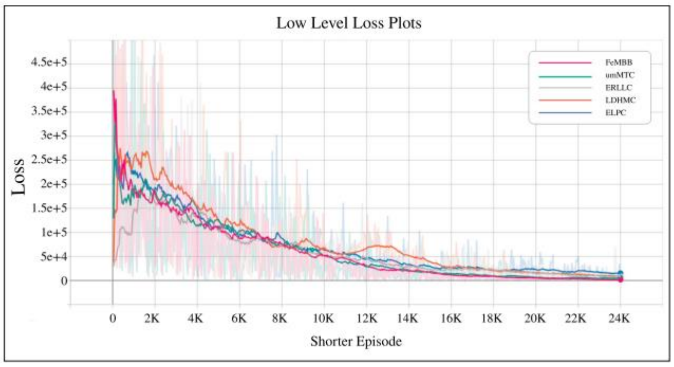


Figure 5 - Losses at low-level part

By jointly considering Figures 4 and 5, we notice that both levels' training processes have succeeded. However, since our proposed ML techniques works in a network environment, we need to ensure that the agents can manage and allocate network resources in the system. Due to the fact that there are no similar research works in the literature to use as the baseline and compare the results, we used the trained model in the same interactive environment to test some KPIs and their value. This test monitors three of the network KPIs in 1000 episodes to assure that KPIs are kept in the predefined range by the agents. If so, the purpose of this work has been fulfilled. As the system level and user-experienced data rates are the two critical KPIs in the FeMMB service type, we chose these two KPIs to monitor in the environment using the trained agents. In addition, latency was observed as a critical KPI in the ERLLC service type. According to Figure 6, the system peak data rate has been kept at more than 1 Tbps in all 1000 episodes. Based on the user-experienced data rate, a minimum of 1 Gbps data rate has been provided. The uniform distribution without having any sparse large value among the numerical results show that the upper limitations to control unnecessary assignments in the decentralized part have guaranteed the optimal assignments. Finally, Figure 6 shows that latency in the ERLLC service type is less than 10 µs in all episodes, confirming the success of both levels' agents' performance. Having the desired value for the network KPIs, as shown in Figure 6, affirms that agents in both levels perform the expected duties, and the overall solution works smoothly and can be a promising technique for the 6G complex system model.

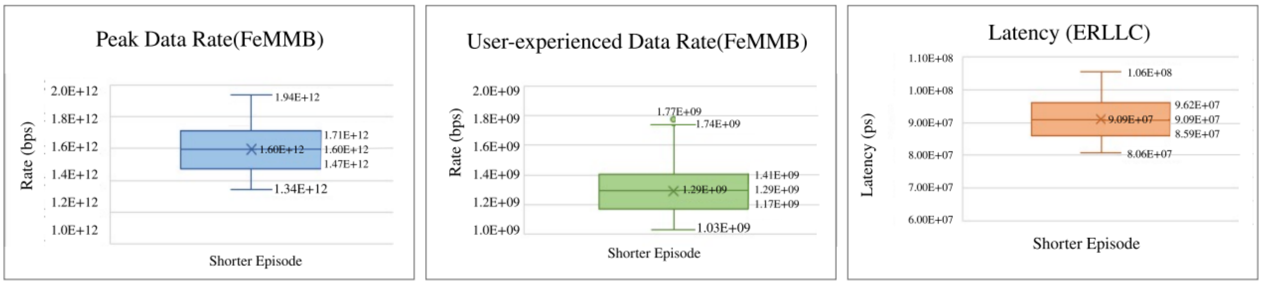


Figure 6 - Network KPIs: (a) Peak data rate, (b) User-experienced data rate, (c) Latency

2.4 Conclusions and Future work

The intelligent information society and emerging applications in the early future demand the next generation of wireless networks to overcome current limitations and provide different quality and service levels. New technologies that introduce openness, flexibility, and intelligence to the network are needed to address these demands. This paper first details candidate technologies for 6G system models such as O-RAN and then proposes a two-level DRL-based network slicing and resource assignment, which is compatible with these new system models and aims to maximize the network capacity while providing QoS of each service type. The proposed scheme benefits from imitating human teamwork and offers the flexibility and intelligence of agents that can observe, learn, and take actions online. Having two levels of performance and time not only facilitates optimal assignments but also helps the complex nature of the problem to become practical and scalable. Moreover, to study the performance of the scheme, Deep ML algorithms are proposed for the 6G environment at both levels and implementation details are discussed. The choice of algorithms at each level has been based on compatibility and complexity concerns. Since this work has considered a new system model, the compatible environment was simulated using the OpenAI Gym library; the general idea of managing the slices and assigning resources can be used with other proper algorithms and network environments.

Even though one of the objectives of this approach is to address the complex problem of network slicing and resource management in a system model consisting of CF in RAN, in this research work, low-level agents only try to provide the required KPIs by assigning resource blocks. On the other hand, KPIs such as mobility and latency are affected significantly by communication links and physical layer-related management. The lower part can be improved to consider both communication links and resources, in addition to creating clusters for users in the CF RAN. Moreover, in our current simulations, each agent is trained for one service type. Having agents that can be trained for all types can robust the management. Adopting other algorithms to each level and simulating different environments may result in exciting results.

3 Framework: Clustering-Driven Approach for Base Station Parameter Optimization and Automation (CeDA-BatOp)

The relentless growth of mobile data traffic and the rapid evolution of mobile communication technologies have imposed immense pressure on network infrastructure. Base station parameters, the cornerstone of network performance, demand meticulous optimization and management to ensure efficient and reliable operation [8]. Traditional methods for optimizing base station parameters are often manual, time-consuming, and prone to errors, necessitating the development of novel machine learning-based approaches that can automate and optimize base station parameter management [9].

This comprehensive analysis delves into two proposed novel methodologies for optimizing and managing base station parameters in mobile communication networks: CEDA-BatOp and CEDA-BatOp 2.0. These methodologies introduce innovative machine learning techniques to optimize base station parameters and address parameter drift, paving the way for more efficient and reliable mobile communication networks.

The contribution section outlines the substantial advancements and novel contributions brought forth by the research endeavor. At its core, this work introduces a comprehensive framework designed for base station parameter optimization in cellular networks. It presents a pioneering approach that amalgamates machine learning methodologies, specifically clustering techniques, with the aim of enhancing network performance. The key contribution lies in the development of a versatile and adaptable framework capable of optimizing base station parameters across diverse network scenarios. By addressing the limitations observed in existing solutions, particularly their applicability to specific network problems or settings, this framework emerges as a solution offering broader applicability. The intricate details of this framework, involving a three-stage process encompassing training, fine-tuning, and drift monitoring, establish a methodical and efficient approach towards optimizing base station parameters. The framework's capacity to adapt to varying parameter types and different network conditions underscores its versatility. Additionally, the study delves into drift monitoring within cellular networks, a facet relatively underexplored. By integrating mechanisms to detect and address data drift, the framework ensures the adaptability and robustness necessary for optimal network operation. Overall, the significant contribution lies in not only devising a comprehensive optimization framework but also in extending its scope towards addressing pertinent issues like data drift, thus marking a substantial advancement in network optimization and adaptability.

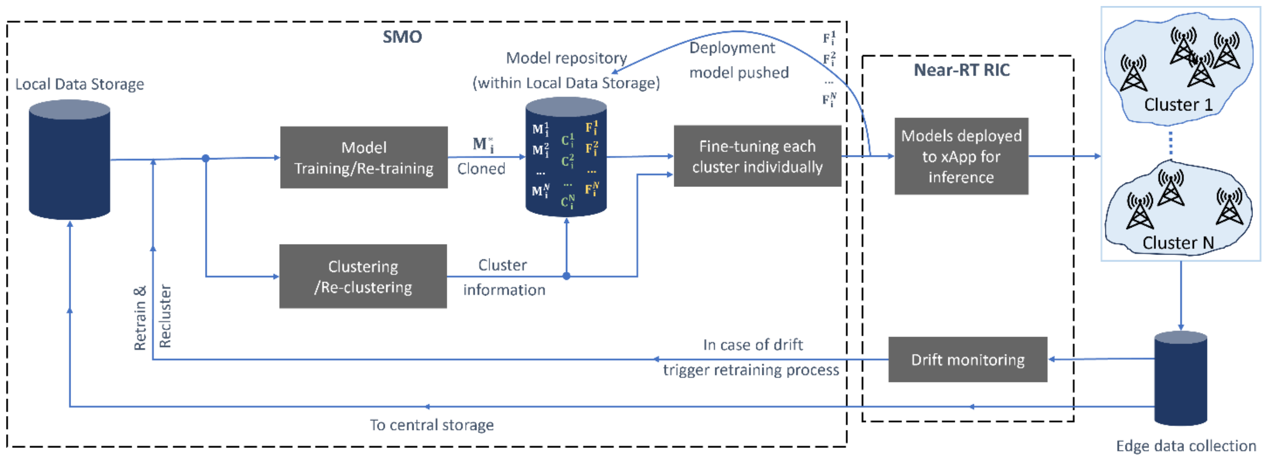

SMO
Local Data Storage
Model repository (within Local Data Storage)
Deployment model pushed
Model Training/Re-training
Cloned
Clustering /Re-clustering
Cluster information
Fine-tuning each cluster individually
Near-RT RIC
Models deployed to xApp for inference
Cluster 1
Cluster N
Retrain & Recluster
In case of drift trigger retraining process
Drift monitoring
To central storage
Edge data collection

Figure 7 - Base station Parameter Optimization and Automation Framework

The framework is structured into three key stages, each contributing to the overarching optimization process.

Stage 1 (Training and Clustering): The initial phase involves training a base model (M∗1) on the offline dataset D1 and clustering base stations in parallel. Clustering, performed via pairwise constrained clustering techniques, aims to create data clusters (C11...CN1) for the initial iteration. Models (M11...MN1) are then created for each cluster by cloning the pre-trained base model. This stage establishes a baseline for comparison and prepares for subsequent retraining cycles. Retraining involves obtaining new models (M1i...MNi) by training the pre-trained models on updated datasets (Di) obtained from UEs, concurrently with base station clustering based on the new data.

Stage 2 (Fine-Tuning and Prediction): Post Stage 1, the data clusters (C1i...CNi) from each iteration undergo fine-tuning to optimize their respective pre-trained models (M1i...MNi). The fine-tuned models are then deployed to infer base station parameters via the xApp [10], contributing to enhanced accuracy and specialized predictions tailored to each cluster.

Stage 3 (Drift Monitoring and Retraining): Continual monitoring of base station data for data drift is pivotal in maintaining model accuracy. In the event of detected drift, the framework initiates the retraining process using new UE data (oi) while assuming base station labels are obtainable from a lookup table. This proactive approach, requiring minimal human intervention, ensures adaptive model updates in response to changing network conditions.

The left and right dotted lines in the framework diagram delineate components deployable in Service Management & Orchestrator (SMO) and Near-RT RIC, respectively. This modular setup enables efficient deployment and integration of the framework components into network management systems, facilitating seamless optimization and automation in cellular networks.

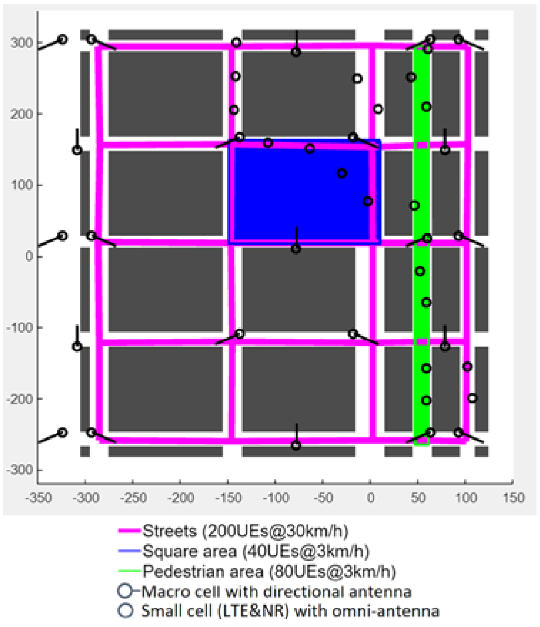

Streets (200UEs@30km/h)
Square area (40UEs@3km/h)
Pedestrian area (80UEs@3km/h)
Macro cell with directional antenna
Small cell (LTE&NR) with omni-antenna

Figure 8 - Map of a simulated locality in Madrid

Simulation Setup: The simulation commenced with the creation of an initial dataset employing a dynamic system level simulator replicating a Madrid locality scenario. This simulation encompassed 42 base stations mirroring an urban environment—a mix of macro cells with directional antennas and small cells equipped with Omni-antennas. The scenario incorporated diverse user mobility patterns, involving UEs moving at varying speeds (200 UEs at 30 km/hr, 40 UEs at 3 km/hr, and 80 UEs at 3 km/hr). The dataset encompassed an array of UE parameters, including RSRQ, RSRP, RSSI, SINR, CQI, DL/UL bitrate, along with parameters from neighboring base stations. Similarly, optimized base station parameters such as antenna tilt, azimuth, and maximum transmission power were part of this comprehensive dataset. This diverse setup was designed to reflect realistic urban scenarios and offer a robust foundation for evaluating base station-specific machine learning models.

Discussion and Result: The evaluation process involved a wide array of machine learning models, ranging from traditional approaches like SVM Regressor, SGDRegressor, to more advanced techniques such as Gaussian Process Regression (GPR) and Feedforward Neural Networks (FCNN) of varying depths. The aim was to optimize these models for base station parameter prediction. A meticulous hyperparameter tuning phase ensued, aiming to fine-tune these models to the complexities inherent in the dataset. Metrics such as Root Mean Square Error (RMSE), total training time, and storage requirements were analyzed to comprehensively understand each model's performance in optimizing base station parameters.

Table 2 - Comparison of clustering algorithms (K-means, DB-SCAN, AHC)

Clustering Algorithm	Avg. Silhouette Score	Avg. Time Taken
K-means	0.527	∼11 mins
DBSCAN	0.596	∼13 mins
AHC	**0.779**	∼23 mins

Table 3 - Performance of models when trained on the overall dataset without fine-tuning

Model	RMSE	Total Training Time (avg.)	Total storage occupancy (MB)
SVM Regressor – RBF Kernel	46.57	∼1hr 43 mins	∼11
SGDRegressor	37.33	∼2hr 32 mins	∼7
GPR - DotProduct() + WhiteKernel(noise_level=0.5)	31.32	∼4hr 37 mins	∼17
GPR - Exponentiation(RationalQuadratic(), exponent=2)	24.43	∼5hr 28 mins	∼19
GPR - RBF() + ConstantKernel(constant_value=2)	37.38	∼4hr 29 mins	∼18
FCNN – 4 layers (12,25,10)	44.43	∼2hr 13 mins	∼11
FCNN – 5 layers (12,25,12,10)	**28.34**	∼2hr 49 mins	∼14
FCNN – 6 layers (12,25,25,12,10)	**27.23**	∼3hr 23 mins	∼24

Table 4 - Performance of models when trained on the overall dataset with fine-tuning

Model	RMSE	Total Training Time (avg.)	Total storage occupancy (MB)
SVM Regressor – RBF Kernel	38.23	∼1hr 48 mins	∼21
SGDRegressor	32.73	∼2hr 39 mins	∼24
GPR - DotProduct() + WhiteKernel(noise_level=0.5)	19.74	∼4hr 42mins	∼32
GPR - Exponentiation(RationalQuadratic(), exponent=2)	17.32	∼5hr 34mins	∼44
GPR - RBF() + ConstantKernel(constant_value=2)	17.92	∼4hr 41mins	∼51
FCNN – 4 layers (12,25,10)	37.03	∼2hr 22 mins	∼81
FCNN – 5 layers (12,25,12,10)	**16.98**	∼3hr 01 mins	∼111
FCNN – 6 layers (12,25,25,12,10)	**15.08**	∼3hr 46 mins	∼147

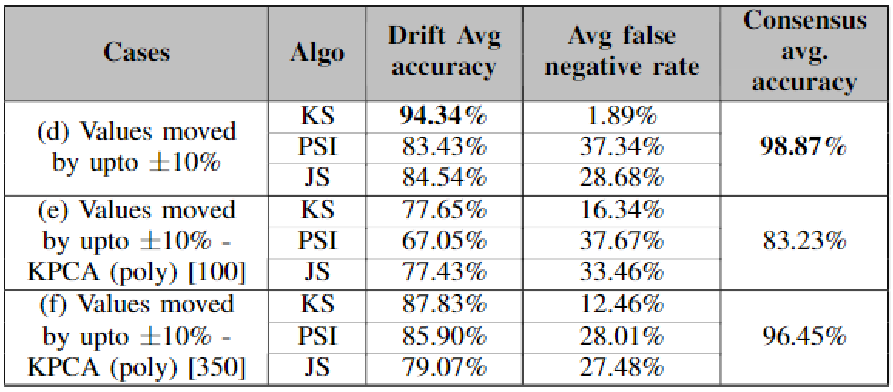
Table 5 - Performance of FCNN - 5 layers (12,25,12,10) average of 42 models

Model	RMSE	Total Training Time (avg.)	Total storage occupancy (MB)
FCNN – 5 layers (12,25,12,10)	14.34	∼3hr 46 mins	∼466

Table 6 - For data moved by up to ±5

Cases	Algo	Drift Avg accuracy	Avg false negative rate	Consensus avg. accuracy
(a) Values moved by upto ±5%	KS	**90.12%**	1.57%	**96.43%**
	PSI	73.64%	39.35%	
	JS	84.32%	27.11%	
(b) Values moved by upto ±5% - KPCA (poly) [100]	KS	72.77%	13.54%	82.30%
	PSI	59.53%	46.21%	
	JS	76.34%	35.54%	
(c) Values moved by upto ±5% - KPCA (poly) [350]	KS	85.43%	9.32%	93.53%
	PSI	78.32%	42.64%	
	JS	78.11%	30.34%	

Table 7 - For data moved by up to ±10



Cases	Algo	Drift Avg accuracy	Avg false negative rate	Consensus avg. accuracy
(d) Values moved by upto ±10%	KS	**94.34%**	1.89%	**98.87%**
	PSI	83.43%	37.34%	
	JS	84.54%	28.68%	
(e) Values moved by upto ±10% - KPCA (poly) [100]	KS	77.65%	16.34%	83.23%
	PSI	67.05%	37.67%	
	JS	77.43%	33.46%	
(f) Values moved by upto ±10% - KPCA (poly) [350]	KS	87.83%	12.46%	96.45%
	PSI	85.90%	28.01%	
	JS	79.07%	27.48%	

The outcomes showcased the GPR model employing the Exponentiation of RationalQuadratic() kernel as the most efficient, delivering the lowest average RMSE among all models. FCNN models displayed a progressive increase in predictive accuracy as the number of layers increased, albeit with trade-offs in training time and storage requirements. Furthermore, an investigation into individual models for each base station revealed enhanced predictive capability at the expense of increased storage demands, highlighting the potential for improved accuracy by tailoring models to specific base stations.

The section concluded by underscoring the effectiveness of the clustering-driven approach, which facilitated improved network optimization and a substantial reduction in memory overhead. The drift monitoring system's robustness was also highlighted, crucial for maintaining accurate predictions amidst dynamic network conditions. Overall, the results emphasized the potential and versatility of the framework in optimizing cellular networks, with future directions focused on innovations like pseudo labeling and exploration of applications beyond optimization to address network disruptions and adapt to evolving traffic patterns.

3.1 CeDA-BatOp 2.0: Enhanced Framework for Base Station Parameter Optimization and Automation with Joint Optimization, Controlled Drift Analysis and Pseudo-Labeling

The contributions of CeDA-BatOp 2.0 mark a significant advancement in the realm of base station clustering and parameter optimization, presenting a trio of pioneering enhancements over its predecessor, CeDA-BatOp 1.0. Foremost among these innovations is the introduction of a cutting-edge Multi-Task Learning (MTL) methodology. This groundbreaking approach revolutionizes the training process by concurrently harnessing the power of the base station parameter predictor (FCNN-5) and the clustering task (AHC) through a shared encoder architecture. This joint training strategy leads to superior performance metrics, evidenced by substantial improvements in clustering precision and prediction accuracy when compared to traditional independent training methods.

A second pivotal contribution lies in the meticulous exploration of controlled drift phenomena within the dataset domain. CeDA-BatOp 2.0 pioneers a comprehensive analysis of drift dynamics utilizing Gaussian Mixture Models (GMMs) . By deliberately introducing controlled noise into the dataset and observing the system's response, this framework offers invaluable insights into the robustness and adaptability of its drift detection mechanisms. This intricate examination unravels the framework's ability to discern subtle variations and adapt to changing data distributions, a critical feature for real-world network data management.

The third major stride forward is the introduction of an innovative Pseudo-Labeling Strategy employing Multi-View Co-Training (MVCT) [11]. This groundbreaking approach harnesses the distinct perspectives offered by two disparate algorithms, FCNN-5 and HGBoost, to generate informative pseudo labels for unlabeled data. By leveraging the complementary insights from these models, the MVCT technique significantly enhances the retraining process, reducing error rates and refining predictions. This novel strategy showcases the potency of amalgamating insights from multiple models to amplify overall performance.

Collectively, these contributions underscore CeDA-BatOp 2.0 as a groundbreaking evolution in network infrastructure management. This framework not only addresses critical challenges in cellular network optimization but also promises to substantially enhance network performance and user experience in urban settings. Future explorations are poised to delve deeper into reinforcement learning techniques for heightened network management and the development of intelligent weighting systems to further enhance predictions derived from multiple perspectives within the MVCT pseudo-labeling system. In essence, CeDA-BatOp 2.0 heralds a new era in cellular network management, offering an adaptive, efficient, and cutting-edge framework for dynamic network optimization.

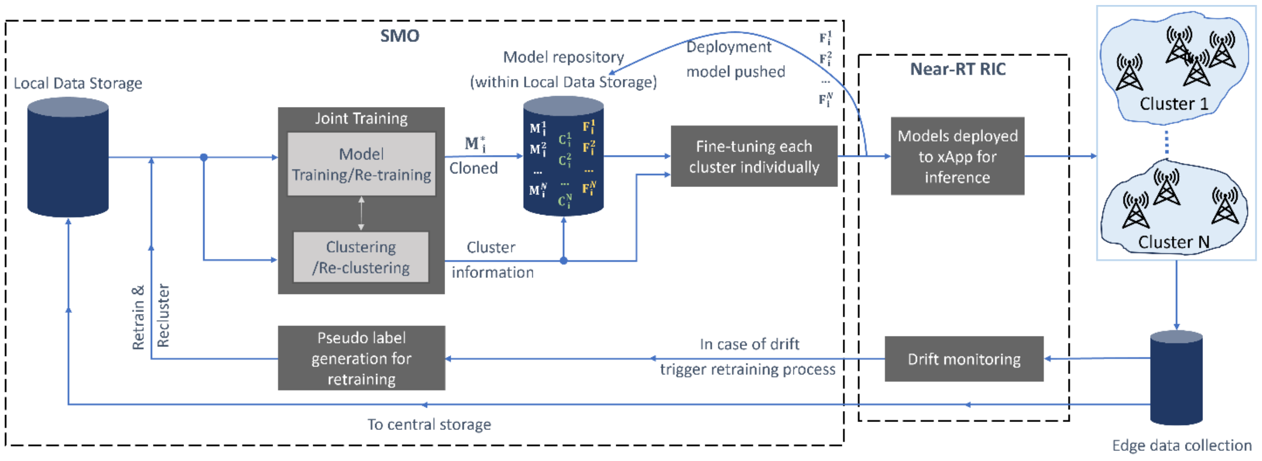

SMO
Local Data Storage
Joint Training
Model Training/Re-training
Clustering /Re-clustering
M_i^*
Cloned
Model repository (within Local Data Storage)
Deployment model pushed
Cluster information
Fine-tuning each cluster individually
Near-RT RIC
Models deployed to xApp for inference
Cluster 1
Cluster N
Retrain &
Recluster
Pseudo label generation for retraining
In case of drift trigger retraining process
Drift monitoring
To central storage
Edge data collection

Figure 9 - Base station Parameter Optimization and Automation Framework

The CeDA-BatOp 2.0 framework embodies a pioneering approach to base station clustering and parameter optimization, comprising a meticulously structured methodology that encompasses distinct stages. This multifaceted framework commences with the creation of an offline dataset (D1) meticulously generated using an in-house simulator and comprising User Equipments (UEs) and base station parameters. Acting as the foundational input and output for the Machine Learning (ML) model, this dataset forms the cornerstone of subsequent stages. The framework delineates four primary stages, commencing with

Stage 1 (Joint Training of Parameter Predictor and Clustering): This phase leverages a shared encoder architecture that unifies the base station parameter predictor (FCNN-5 layers) and the Agglomerative Hierarchical Clustering (AHC) task. The joint training process integrates individual task losses via a methodology inspired by previous works. This strategic fusion not only facilitates the collective learning of features beneficial for all tasks but also enables task-specific adaptations, optimizing the model's overall performance.

Stage 2 (Fine-tuning and Prediction): capitalizes on the cluster information garnered from joint training (C1i to CNi). Here, the pre-trained models undergo a fine-tuning process, refining the models for specific clusters to enhance predictive accuracy. The resulting fine-tuned models are subsequently deployed as xApps, enabling the inference of optimized base station parameters.

Stage 3 (Controlled Drift Analysis and Monitoring): where data acquisition from UEs undergoes continuous monitoring to detect data drift. The framework employs a controlled drift study, employing Gaussian Mixture Models (GMMs) to simulate drift phenomena and meticulously assess its implications within the dataset domain. This stage is pivotal in elucidating the system's adaptability to changing data distributions [12].

Stage 4 (encompasses Pseudo-Label Generation for Retraining): an innovative strategy that utilizes Multi-View Co-Training (MVCT). Employing two views—FCNN-5 and HGBoost—the framework harnesses distinct perspectives to generate pseudo labels for unlabeled data. This collaborative approach between models streamlines the retraining process, enhancing model robustness and accuracy.

Collectively, these stages form a comprehensive framework, setting the stage for CeDA-BatOp 2.0 as an adaptive, efficient, and innovative system. Each phase plays a crucial role in shaping the framework's capabilities, enabling it to address real-world network challenges, and continually adapt to the dynamic nature of cellular network data.

Simulation setup, discussion, and results: This section presents an extensive overview of the carefully curated simulated dataset and the subsequent analyses conducted within the CeDA-BatOp 2.0 framework. Leveraging a proprietary dynamic system-level simulator, the framework employed a meticulously crafted dataset comprising 42 strategically positioned base stations, meticulously mirroring an urban setting in Madrid. This dataset encapsulated a diverse array of base station configurations, including macro cells with directional antennas and small cells utilizing Omni-antennas. Simulated mobility patterns of User Equipments (UEs) further enriched the dataset, incorporating varying speeds across 200 UEs at 30 km/hr, 40 UEs at 3 km/hr, and 80 UEs at 3 km/hr. This diversity aimed to simulate realistic mobility scenarios, enabling the Machine Learning (ML) model to adapt to a spectrum of user behaviors and network configurations.

The dataset itself was a rich repository of UE parameters and optimized base station configurations, encompassing parameters such as Reference Signal Received Quality (RSRQ), Reference Signal Received Power (RSRP), Signal-to-interference-plus-noise ratio (SINR), and base station parameters like Antenna tilt, Antenna azimuth, and Maximum Transmission power. Comprising a vast array of input features at the UE end and output features at the base station end, this dataset laid the foundation for robust analysis within the CeDA-BatOp 2.0 framework.

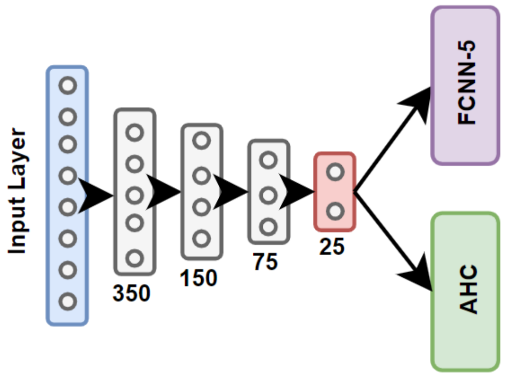


Figure 10 - Joint Training of FCNN-5 and AHC along with the shared encoder

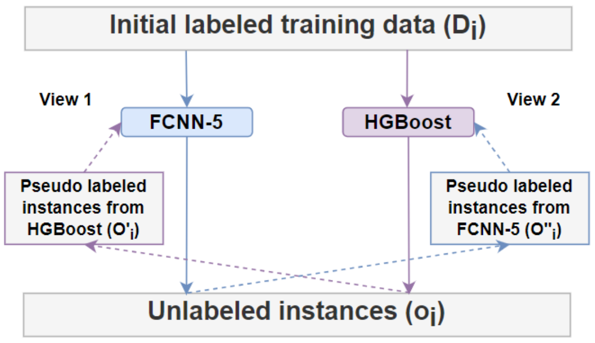

Initial labeled training data (D_j)
View 1
FCNN-5
HGBoost
View 2
Pseudo labeled instances from HGBoost (O'_j)
Pseudo labeled instances from FCNN-5 (O''_j)
Unlabeled instances (o_j)

Figure 11 - Multi-View Co-training: FCNN-5 as view 1 and HGBoost as view 2

The study investigated the framework's efficacy across various stages. Stage 1 involved joint training of the base station parameter predictor and clustering tasks, demonstrating improved performance in both clustering and prediction tasks compared to the previous version, CeDA-BatOp 1.0 [13]. The discussion elaborated on the efficacy of the joint training approach, emphasizing the model's ability to collectively achieve superior solutions by integrating losses from individual tasks.

Furthermore, Stage 2 involved fine-tuning the models based on the clusters obtained in the joint training phase. This stage highlighted significant reductions in Root Mean Square Error (RMSE), showcasing enhanced prediction accuracy, particularly in scenarios where the pre-trained models underwent fine-tuning, signifying the pivotal role of fine-tuning in optimizing predictions for specific clusters.

Additionally, the section detailed the exploration of controlled drift analysis utilizing Gaussian Mixture Models (GMMs). Through meticulous drift experiments, the study assessed the framework's ability to detect and adapt to drift phenomena within the dataset. The discussion emphasized the challenges faced in identifying subtle drift instances and the framework's performance in accommodating unpredictable variations in real-world network data.

Moreover, the exploration of a pseudo-labeling strategy using Multi-View Co-Training (MVCT) demonstrated the efficacy of generating labels for unlabeled data, significantly reducing the need for manual intervention and enhancing predictive accuracy. The discussion highlighted the benefits of leveraging distinct perspectives from FCNN-5 and HGBoost models, showcasing the potential synergy between models to streamline the retraining process.

Overall, this comprehensive analysis underscored the CeDA-BatOp 2.0 framework's adaptability, efficiency, and robustness in addressing real-world network challenges, paving the way for an autonomously functioning network infrastructure.

Table 8 - AHC Comparison for CeDa-BatOp versions 1.0 and 2.0

Clustering Algorithm	Version	Avg. Silhouette Score
AHC	CeDA-BatOp 1.0	0.779
	CeDA-BatOp 2.0	**0.853**

Table 9 - Performance of FCNN-5 (12,25,12,10) with initial training (without cluster fine-tuning)

Model	Version	Initial Training	RMSE
FCNN-5 (12,25,12,10)	CeDA-BatOp 1.0	Not Applicable	28.34
	CeDA-BatOp 2.0	No	19.89
		Yes	**17.62**

Table 10 - Performance of FCNN-5 (12,25,12,10) with initial training and cluster fine-tuning

Model	Version	Initial Training	RMSE
FCNN-5 (12,25,12,10)	CeDA-BatOp 1.0	Not Applicable	16.98
	CeDA-BatOp 2.0	No	12.75
		Yes	**12.34**

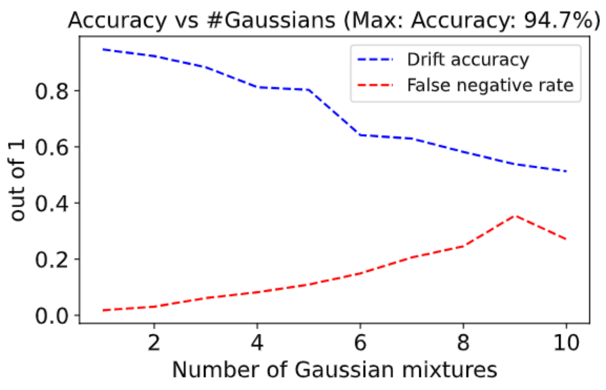


Figure 12 - Drift detection accuracy across GMMs for SNR = 166%

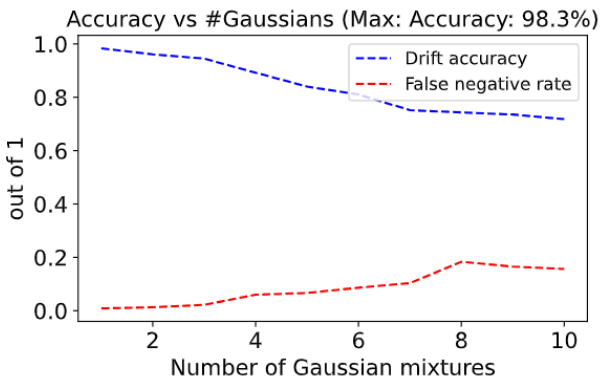


Figure 13 - Drift detection accuracy across GMMs for SNR = 125%

The conclusion of the study highlights CeDA-BatOp 2.0 as a significant advancement in the domain of base station clustering and parameter optimization. This updated framework introduces key improvements over its predecessor, CeDA-BatOp 1.0, primarily through the implementation of multi-task learning, controlled drift analysis, and a strategic pseudo-labeling strategy using Multi-View Co-Training (MVCT). The evaluation of CeDA-BatOp 2.0 on a meticulously curated simulated dataset showcased superior performance in both clustering and prediction tasks, with notable enhancements such as a Silhouette score of 0.853 and an RMSE of 12.34. The discussion emphasized the efficacy of joint training, fine-tuning, and the system's adaptability to controlled data drift, showcasing its resilience in adjusting to ever-changing real-world network data.

The controlled drift analysis served as a critical component, offering insights into the framework's ability to detect and respond to drift phenomena within the initial dataset domain. The exploration underscored the challenges faced in identifying subtle variations and the framework's adaptability to unpredictable changes, essential for dynamic network adjustments. Additionally, the integration of pseudo-labeling via MVCT demonstrated remarkable potential in generating accurate labels for unlabeled data, reducing manual intervention, and enhancing model adaptability.

CeDA-BatOp 2.0's ability to evolve into an autonomously functioning network infrastructure stands as a promising achievement. Its capacity to continuously update models, respond to changing network conditions, and streamline the retraining process lays a robust foundation for dynamic network management. The conclusion also outlines avenues for future exploration, emphasizing an interest in reinforcement learning techniques and intelligent weighting systems to further refine predictions derived from the MVCT pseudo-labeling strategy. In summary, CeDA-BatOp 2.0 presents a clear trajectory toward establishing a more adaptive and efficient network infrastructure, promising significant advancements in dynamic network management and optimization.

3.2 Towards Robust Anomaly Detection in User Equipment Parameters: A Deep Generalized Canonical Correlation Analysis approach

This section emphasizes the significant advancements and novel contributions made through the proposed anomaly detection system using Deep Generalized Canonical Correlation Analysis (DGCCA) [14] for User Equipment (UE) parameters in mobile communication networks transitioning towards 5G.

The core contribution lies in the development of a robust anomaly detection methodology tailored specifically for UE parameters. We introduce UE anomaly detection using DGCCA, an innovative extension amalgamating the strengths of Deep Canonical Correlation Analysis (DCCA) and Generalized Canonical Correlation Analysis (GCCA) [15]. This novel amalgamation offers a unique solution capable of handling multi-view, high-dimensional, and non-linear UE parameter data, effectively surpassing the limitations of traditional anomaly detection models.

Anomaly Detection in UE Parameters Using DGCCA: This section unveils a sophisticated approach rooted in the concept of Canonical Correlation Analysis (CCA) [16] and its evolution into Deep Generalized Canonical Correlation Analysis (DGCCA) for robust anomaly detection within User Equipment (UE) parameters. Commencing with an exposition on CCA, its capability in identifying linearly correlated projections between two vectors is elucidated. However, they expound on its limitations, particularly its restriction to linear projections and its inability to handle correlations among more than two input features. This prompts the introduction of DCCA, a breakthrough approach overcoming the limitations by integrating stacked deep neural networks for each feature and performing CCA on the produced outputs. Despite this advancement, the constraint of correlating only two views persists, leading to the introduction of GCCA, a generalized extension allowing for shared representations among multiple views but still confined to linear projections.

The innovation of DGCCA stems from amalgamating the strengths of DCCA and GCCA, supporting more than two views and non-linear transformations through the expressive prowess of deep neural networks. DGCCA's framework, initially proposed for suggesting friends and hashtags for Twitter users, integrates the training process where input features traverse their respective deep neural networks, and weights are optimized through backpropagation, aligning with the GCCA objective. The architecture for N views in DGCCA is meticulously depicted, primarily based on 1D convolutions followed by convolutional layers, max-pooling, flattening, and feeding into a Fully Connected Neural Network (FCNN).

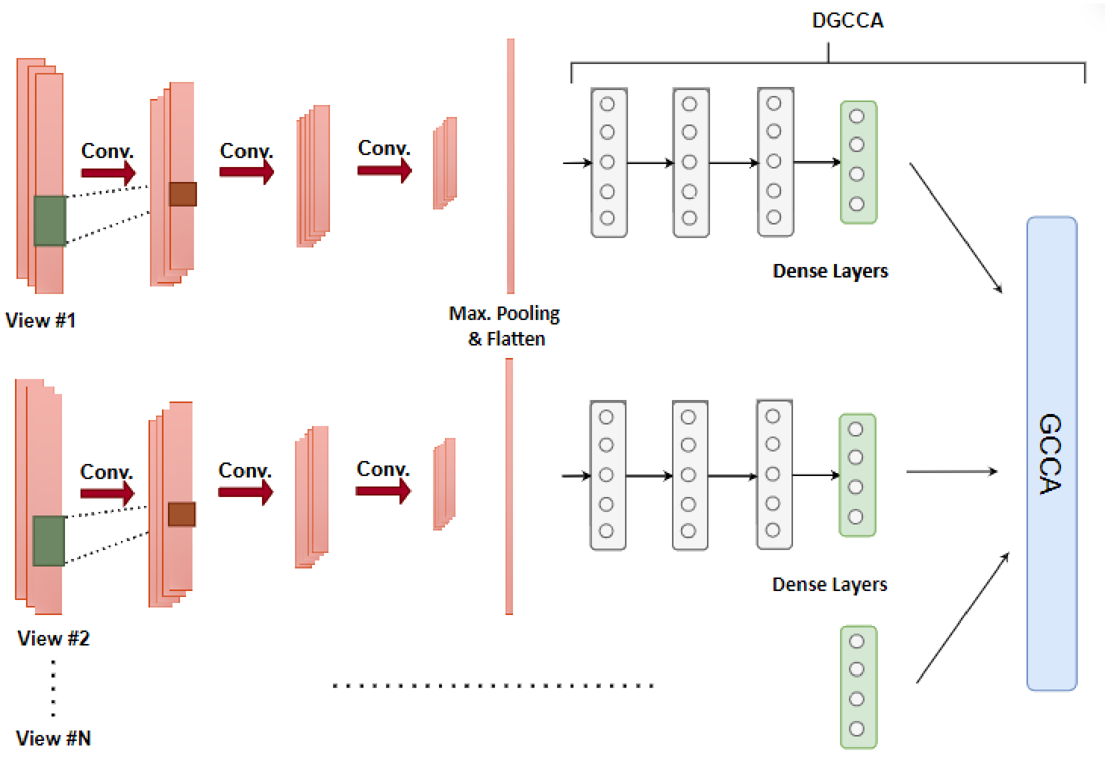


Figure 14 - Simplified Illustration of UE Anomaly detection system for N features (N=5)

The optimization objective of DGCCA involves obtaining a shared representation called 'G' by maximizing the sum of correlation between 'G' and the output of each neural network. This is achieved by computing top 'K' canonical correlations for every pair of networks and summing them to obtain the collective multicorrelation value of N views. Leveraging the correlations between pairs of features, a strategy using threshold values for detecting anomalies within a feature at a certain period is introduced. However, the strategy remains effective only when a single feature is anomalous at a given time, leaving the simultaneous detection of anomalies in multiple features for future exploration. In this context, the section meticulously discusses the determination of hyperparameters, particularly the 10 threshold values obtained from the validation set during training.

The multifaceted approach culminates in the proposal of an intricate and comprehensive anomaly detection system tailored for UE parameters, leveraging the prowess of DGCCA and addressing the limitations of earlier canonical correlation methodologies.

Evaluation and Discussion: This section delves into a meticulous assessment of the proposed anomaly detection system within the context of User Equipment (UE) parameters, using DGCCA, offering a comprehensive insight into the performance metrics, dataset particulars, model architectures, and comparative analysis among various machine learning algorithms. The researchers adeptly employ the CeDA-BatOp v1.0 simulated dataset, meticulously derived from a proprietary dynamic system level simulator, meticulously mirroring an urban landscape in Madrid. This dataset features a diverse array of base station configurations, mimicking realistic UE mobility patterns across different user scenarios with a variety of velocities, enriching the dataset's utility for machine learning model adaptation.

The dataset encompasses critical UE parameters like Reference Signal Received Quality (RSRQ), Reference Signal Received Power (RSRP), Reference Signal Strength Indicator (RSSI), Signal-to-interference-plus-noise ratio (SINR), and Channel Quality Indication (CQI). These parameters form the foundation for the anomaly detection system's training, validation, and testing, rigorously segregated into sets ensuring robust model development and assessment.

Splitting the UE parameter dataset into distinct subsets—70% for training, 15% for validation, and 15% for testing—provides a solid foundation for comprehensive evaluation while ensuring model generalizability. To ensure uniformity and alleviate issues stemming from varied scaling, the data is standardized using PyTorch's transforms.Normalize() [17] based on calculated global mean and standard deviation for respective features, ensuring homogeneity across the dataset.

The detail of the architecture and hyperparameter selection for the 1D Convolutional Neural Network (CNN) encoder and the Fully Connected Neural Network (FCNN) component of the anomaly detection system. They conducted an exhaustive hyperparameter search, optimizing hidden layers, units, and other critical parameters, finally selecting a configuration showcasing optimal validation accuracy across the FCNN layers.

Table 11 - Architecture of 1D CNN in UE parameter Anomaly

Layer	Kernel dims. (HxW)	Output dims. (C@HxW)
Input	-	320@1x1560
1-Conv1d	1x5	160@1x779
2-Conv1d	1x5	80@1x389
3-Conv1d	1x5	40@1x194
4-Maxpool	1x3	40@1x64
5-Flatten	-	2560

The experiment involves the introduction of random noise at varied time intervals to simulate real-world scenarios and evaluate the anomaly detection system's performance. This deliberate introduction of Gaussian Noise across the features—RSRP, RSRQ, RSSI, SINR, CQI—presents a challenging yet realistic scenario for evaluating model robustness.

The section provides a comprehensive comparison of various machine learning models—SVM, Naive Bayes, Random Forests, FCNN, and DGCCA—showcasing their accuracy and false negative rates. The evaluative metrics underscore DGCCA's superiority, exhibiting an outstanding accuracy of 92.23% alongside an exceptionally low false negative rate of 1.55%, positioning it as the optimal choice for UE parameter anomaly detection.

Table 12 - Anomaly detection of UE parameter with different

Algo.	Accuracy	Avg. False Negative Rate
SVM	72.84%	22.43
Naive Bayes	81.49%	15.85
RF	83.33%	8.12
FCNN (256, 512, 128)	86.18%	6.74
DGCCA	**92.23%**	**1.55**

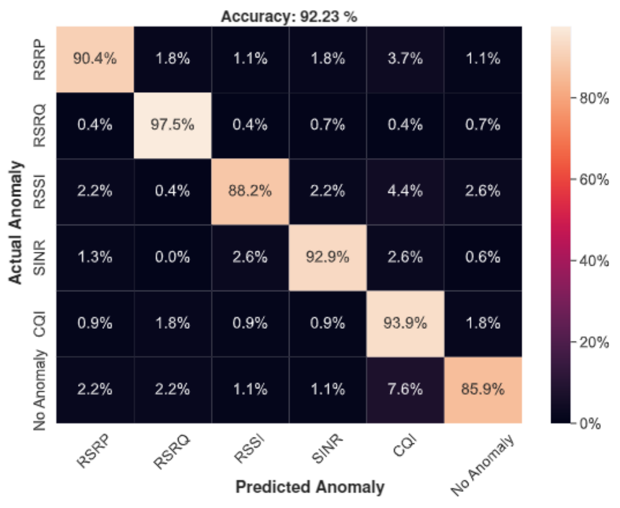


Figure 15 - DGCCA Confusion Matrix

Additionally, we explored the confusion matrix of DGCCA, shedding light on the model's exceptional performance in detecting anomalies within specific parameters like RSRQ while highlighting challenges in discerning non-anomalous instances, especially within the CQI parameter. This critical analysis lays the foundation for future refinements aimed at enhancing the model's capacity to accurately identify anomalies in intricate UE parameters.

Firstly, we highlight the exceptional performance achieved by DGCCA, showcasing an impressive accuracy of 92.23% and an astoundingly low false negative rate of 1.55%. These results validate DGCCA's robustness in discerning anomalies within UE parameters, affirming its superiority over traditional machine learning models like SVM, Naive Bayes, and Random Forests, establishing it as the pinnacle choice for anomaly detection in this domain.

Furthermore, we emphasize DGCCA's pivotal role in enhancing the prediction of optimized base station parameters within the Open Radio Access Network (ORAN) [18] framework. By effectively identifying anomalies in UE parameters, DGCCA contributes significantly to network resilience, adaptability, and overall performance, thereby augmenting self-organizing networks' efficiency and reliability.

The future trajectory of research in this domain, focuses on refining DGCCA's anomaly detection capabilities, especially within complex parameters like CQI. Themphasize the need for further exploration and refinement to address the challenges observed in specific parameters, ensuring DGCCA's applicability and accuracy across diverse network environments.

Additionally, we emphasize the importance of scalability assessment to ascertain DGCCA's effectiveness in real-world scenarios. This avenue of exploration aims to validate DGCCA's utility and performance across varying network landscapes, ensuring its reliability and efficacy in dynamic mobile communication environments.

3.3 Conclusions and Future work

In conclusion, the research demonstrates significant advancements in the domain of base station parameter optimization and anomaly detection within mobile communication networks. CeDA-BatOp 2.0, an updated framework, introduces multi-task learning, controlled drift analysis, and a strategic pseudo-labeling strategy via Multi-View Co-Training (MVCT). Evaluated on a curated dataset, it showcased superior performance in clustering and prediction tasks, affirming its adaptability and resilience in dynamic network environments. Moreover, the Controlled Drift Analysis highlighted the framework's ability to detect and respond to data drift, crucial for network adaptability. The integration of MVCT's pseudo-labeling demonstrated potential in accurately labeling unlabeled data, reducing manual intervention, and enhancing adaptability. DGCCA serves as another pivotal tool, boasting exceptional anomaly detection accuracy and low false negative rates. Its contribution to predicting optimized base station parameters within the Open Radio Access Network (ORAN) framework enhances network resilience and efficiency.

The conclusion outlines avenues for future exploration, emphasizing the refinement of anomaly detection capabilities, especially in complex parameters like CQI, and scalability assessment across diverse network landscapes. These advancements solidify DGCCA's status as a fundamental tool for optimizing self-organizing networks in the evolving landscape of mobile communications.

4 Data-driven Automation for Disturbance Management in Large-scaled Telecommunication Networks

4.1 Introduction

Towards developing a data driven network control and automation platform to achieve the SEMANTIC KPI and target value of 10-fold improvement in re allocating the resources, in Del 4.1, Telenor reviewed data analysis tools, monitoring systems, data sources, and possible use cases based on SoA. Telenor proposed the concept of self-driving and zero touch networks to describe how it intends to use data-driven network control and automation tools in Telenor network. To this aim, in del 4.2, we addressed the optimization of network performance and resource usage by developing a framework based on machine learning algorithms in network disturbance domain toward maximizing Telenor customer satisfaction and minimizing network downtime. We also presented an initial analysis of the system model’s performance in Del 4.2. In Del 4.3, we will give an extensive evaluation of two use cases that we addressed within network disturbance domain based on proposed system model and data sets from real running fixed and mobile Telenor network domains.

Traditional approaches to manage networks are often time and cost-consuming and prone to errors. In fact, the over-time need for continuous connectivity, optimizing the user experience, and dynamic networks such as 5G and beyond necessitates engaging automation in network management processes. Automation helps adjust networks in real-time, efficiently allocate resources, and proactively identify and address network issues before they escalate. Furthermore, the integration of AI and ML has further pushed automation forward by introducing the concept of self-driving networks. Such networks can autonomously monitor, analyse, and optimize their performance, minimizing human intervention [19]. However, legacy systems in most cases, introduce practical challenges for having a fully autonomous network. In our context, we are dealing with a large-scaled real-running complex telecommunication (telco) network, and our goal is to introduce intelligent automation to optimize the management of this network. In fact, rather than automating the design of the network, we aim to explore the application of data-driven approaches for automating the operations of network disturbances. Our goal is to fulfil this task by implementing an automated assistant system on top of our existing network infrastructure reality [20].

At Telenor, a leading telco service provider in Sweden, network disturbances and the actions taken to address them are stored within an internal incident handling system through Trouble Tickets (TTs). Telenor's NOC already employs some levels of automation for handling TTs; however, this automation is mainly based on the predefined rules that human experts have realized through experience over the years. Relying on human expertise could be challenging since it can cause errors and delays [21]. Given the vast historical dataset of TTs, ML techniques can provide an optimized way of automation. Nonetheless, challenges arise due to the heterogeneous temporal information in aggregated TTs, data availability limitations, etc.

Our work addresses challenges related to automating TTs generated from fixed access switches and radio access base stations, in fixed and mobile network domains (We call them fixed and mobile TTs). To this end, we define two use cases: 1. TTs resolution time prediction 2. TTs on-site dispatch-need prediction. To the best of our knowledge, our work is the first to tackle the telco network TTs automation by thoroughly studying their evolution over time using data-driven approaches and presenting comprehensive results. Our study uses the Telenor TT dataset which establishes the developed models to be pragmatic and a step towards automation of TT resolution at Telenor and other telecom service providers with similar set up.

4.2 Background

Consumer services offered by Telenor Sweden include mobile (2.9M subscribers), broadband (700K subscribers) and TV (500K subscribers). Telenor owns a fixed and mobile network consisting of thousands of switches and radio base stations distributed over the country to provide these services.

The fixed network receives incoming data traffic from devices connected to its ports and forwards it to the destination in the Local Area Network (LAN). Access switches connect the devices in LAN to the internet through the core IP network (Figure 16). Access switches provide services such as Voice Over IP (VOIP), fixed broadband, and TV (IPTV, DTV, etc.) to end users. Telenor Sweden owns approximately 35,000 access switches across the country. A mobile network consists of radio base stations. A base station provides a wireless connection between mobile devices such as cell phones, tablets, and other wireless-enabled devices within a designated coverage area, known as a cell (Figure 16). The set of cells creates a Radio Access Network (RAN). Almost 8,000 base station cells across Sweden belong to Telenor.

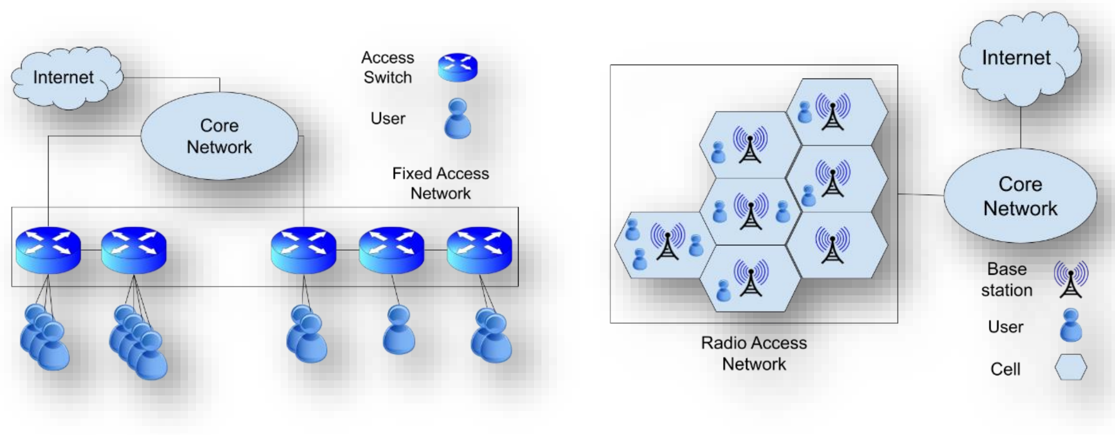
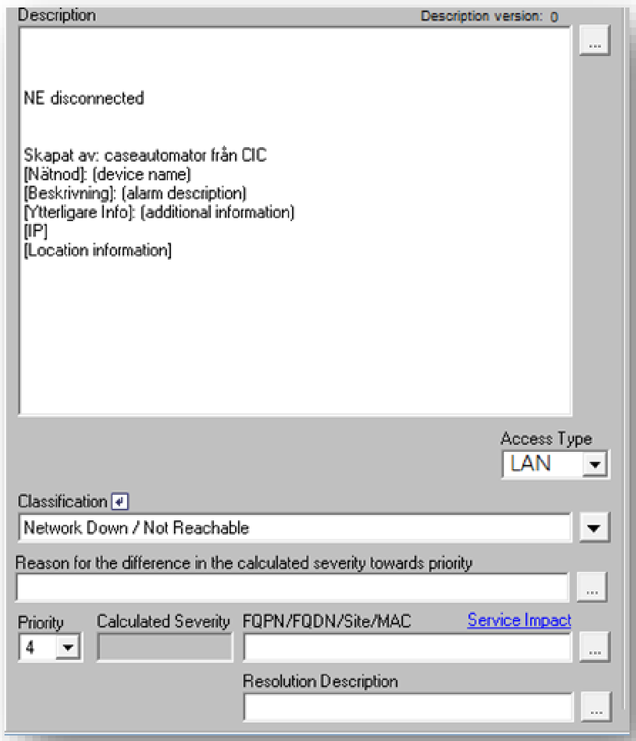


Figure 16 - Fixed Access Network (left), Mobile Access Network (right)

Telenor is managing complex running networks consisting of thousands of interconnected fixed and mobile service-providing elements. Different types of disturbances are expected to happen in these complex networks. A network disturbance happens because of faults in the elements. Timely response to these disturbances can guarantee the smooth operation of the network. Network disturbances are collected in the form of TT records. All TT monitoring and management processes are performed in the incident handling system.

TTs go through a journey from the moment they are created. NOC administrators work on resolving the TTs during this journey and make logs of their actions [22]. At each time stamp of this journey, the values of TT fields might change based on the information realized from the disturbance up to that moment. Part of a TT record is shown in figure 17. Figure 18 shows the diagram of TTs creation, handling, updating, and resolution over time (left) and TT evolution over time (right).


Description Description version: 0

NE disconnected

Skapat av: caseautomator från CIC
[Nätnod]: (device name)
[Beskrivning]: (alarm description)
[Ytterligare Info]: (additional information)
[IP]
[Location information]

Access Type: LAN

Classification: Network Down / Not Reachable

Reason for the difference in the calculated severity towards priority:

Priority: 4
Calculated Severity:
FQPN/FQDN/Site/MAC:
Service Impact

Resolution Description:

Figure 17 - Part of a TT record

When a TT is resolved, the period it took for resolution (resolution time) and information about the necessity of dispatching workforce to the site (dispatch need) are recorded in separate fields in the TT record. Currently, the company reports 480 minutes resolution time for all TTs once they are generated from access switches and radio base stations [22,23]. This is an estimation based on the expertise and service level agreement. However, there is no estimation for the dispatch need. It is usually realized by monitoring the disturbance symptoms during the TT handling procedures. This could take a few minutes to several hours, depending on the problem. This paper investigates predicting the resolution time and dispatch need to facilitate NOC operations.

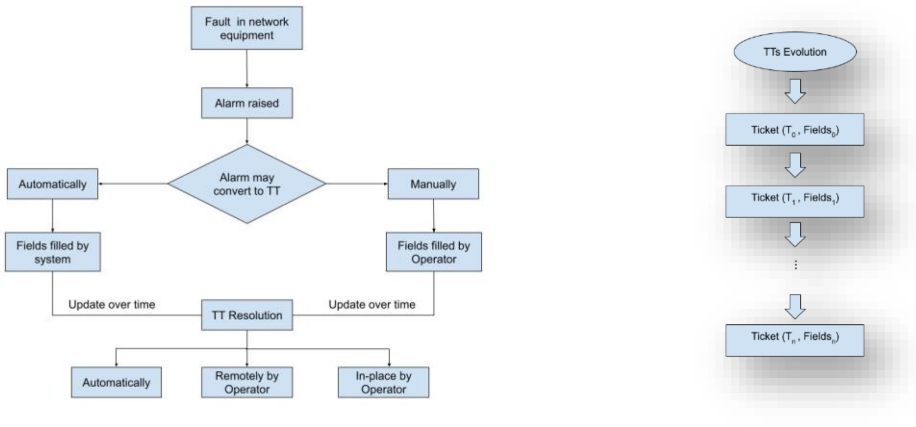


Figure 18 - TT creation, handling, updating, and resolution (left) and TT evolution (right)

4.3 Data Analysis and Predictive Models

This study uses approximately 40,000 switch TTs and 22,000 mobile TTs. We analyze these two datasets separately because of their different fields and characteristics. In this study, we only consider the fields with structured formats, ignoring those filled with human natural language. Using regular expressions, we manage to mine the textual fields of the TTs and extract vital information about disturbances. We refer to the fields and extracted information as features of the TT. Not all features are available at the time of TT creation (T0). They are added or updated overtime as TTs evolve (T1, T2…). There are many blank fields in both datasets. To prepare data for model building, we fill out the categorical blank fields with a constant value and infer blank numerical fields from known parts of the data using the iterative imputation technique [24]. Furthermore, we encode the categorical features so that '1' stands for the category that appears in the field and '0' stands for the remaining categories. We also scale the numerical features between 0 and 1 to speed up the optimization process of the models' cost function.

Figures 19 shows the histogram of the switch and mobile TTs resolution time distribution. Based on this figure, most TTs are resolved within 480 minutes (8 hours) of their creation. All resolved TTs in over 1000 minutes are aggregated in the last bar. There are TTs with unusually long resolution times in both data sets. We consider them outliers and do not consider them in analysis. After removing the outliers and discussing with experts, the range of reasonable resolution time that we consider for this study is 1 to 4,320 minutes, equivalent to 3 days. Figure 20 shows the distribution of dispatch need for switch and mobile TTs. In this case, the target feature is binary (Yes or No). In both data sets, almost 11% to 12% of the TTs need on-site work. This indicates an imbalanced distribution in the data sets for predicting dispatch needs. Finally, after taking all data engineering and cleaning steps, we are left with almost 39,000 switch TTs and 21,000 mobile TTs, each having 24 and 71 different features, respectively.

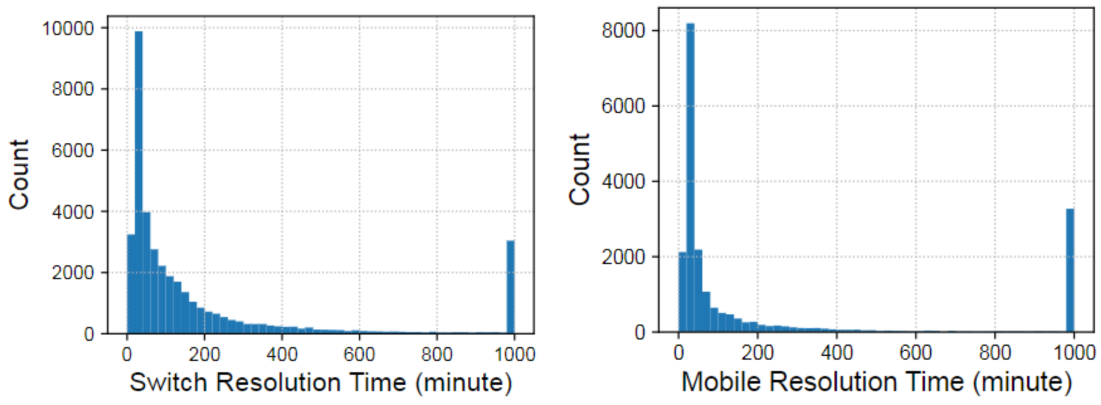


Figure 19 - Distribution of TTs resolution time, Switch (left) Mobile (right)

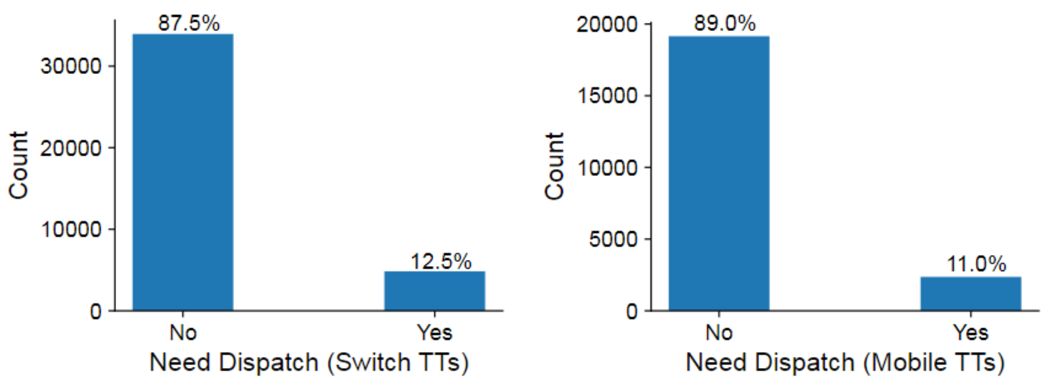


Figure 20 - Distribution of TTs' dispatch need, switch (left), mobile (right)

We want to build models to predict the resolution time and dispatch need for switch and mobile TTs. We train the models on the set of TTs' features available at the time of ticket creation; the target features for prediction are TTs' resolution time and TTs' dispatch need. We keep 80% of the data for training the models and 20% for testing their performance on unseen data. Since the target feature is continuous, we use ML regression models in the resolution time prediction. The regression models that we used for this prediction are as follows:

- Linear Regression (LR) [25]
- K Nearest Neighbour (KNN) [26]
- Decision Tree (DT) [27]
- Random Forests (RF) [28]
- Extreme Gradient Boosting (XGB) [29]
- Neural Network (NN) [34]

We use 3-fold cross-validation and Bayesian search CV [30] for tuning the hyper-parameters and optimizing the results on Mean Absolute Error (MAE) metrics. Using Bayesian search CV, we can efficiently explore the search space of the parameters. To find the best parameter set, we iterate on the parameter space of each model 100 times. Table 13 shows the parameter space we consider for each model.

Table 13 - Regressors' parameter space

Model	**Parameter space**
KNN	n_neighbors: [2,15]
LR	fit_intercept:[True, False]
DT	max_depth:[10,800], max_features:[0.5,1]
RF	max_depth:[1,100], n_estimators:[100,1000], max_features:[0.5,1], max_samples: [0.5,0.99], bootstrap:[True,False]
XGB	learning_rate: [0.01,0.1], max_depth: [1,100], n_estimators: [100,1000], colsample_bytree: [0.5,1] subsample: [0.5,1]
NN	nodes:(64,128,256), batch_size: (4,8,32,64), optimizer: (rmsprop, adam, nadam, sgd), activation_func:(relu, tanh, relu)

In the case of dispatch need prediction, we apply ML classification models. The classifiers that we used for this task are as follows:

- Logistic Regression (LR)
- Random Forest (RF)
- Extreme Gradient Boosting (XGB)

Five-fold cross-validation and randomized search CV [31] are used to optimize the results on the Area Under the Curve (AUC) score. Five different metrics, namely F1 score, precision, recall, False Positive (FP), and False Negative (FN) percentages, are reported for this task. Table 14 shows the parameter space we consider for each model.

Table 14 - Classifiers' parameter space

Model	Parameter space
LR	C:{10−4,10−3,10−2,10−1,10,101,102,103,104}, Penalty:{l1, l2}
RF	max_depth: [2, 14], n_estimators:{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, min_samples_split:[2,12], max_samples_leaf: [1, 11]
XGB	learning_rate: {0.007, 0.008, 0.009, 0.01, 0.02, 0.03}, max_depth: [3,12], n_estimators:{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, colsample_bytree: {0.1, 0.3, 0.5, 1} subsample: {0.1, 0.3, 0.5, 1}

KNN considers the distances in space to learn the similarity (feature-wise) among samples [32]. LR in regression and classification cases is designed to find the linear relationship between independent and target features [33]. Tree-based models also tend to find the non-linearity by building one or multiple decision trees. In DT, only one tree-like model of decisions is used to predict the target [32]. On the other hand, RT and XGB are ensemble models, combining the results of multiple decision trees to reach a more potent effect. They are different in the way they build and combine the trees. In RF, a technique called bagging is used to create decision trees; however, in XBG, the gradient boosting algorithm over an objective function is leveraged to model the process [32]. Relationships in NN are learned using neurons and the connections among them [34].

4.4 Numerical Evaluation

First, we build the models based on all information available at the creation time of TTs (T0). We do this to inform NOC administrators and customers about the probable time ranges within which the disturbances will be resolved. In switch TTs, the disturbance information is updated over time. In mobile TTs, the way the data is collected about disturbances differs from switch TTs, and there is no such updating of information over time. We train the models on the train set and evaluated them on the test set after tuning their parameters. Figure 21 shows the models' MAE results on switch and mobile train, validation, and test sets at T0. For switch TTs, all models perform better than the company baseline with MAE of 430 minutes on the test set. LR and KNN with MAE of almost 240 minutes on the test set show weak performance in predicting the target. Instead, tree-based models such as DT, RF, XGB, and NN have good performance for this prediction. XGB and NN with MAE of around 85 minutes on the test set perform almost the same, outperforming other models. This indicates the non-linearity relation among the switch TTs features. For mobile TTs, all models perform better than the baseline (MAE around 480 minutes on test).

KNN perform better than LR and DT. This shows that there is a similar pattern among mobile TTs. DT among tree-based algorithms has the worst performance. This is probably because of a low number of model parameters used by this algorithm for finding the non-linearity in data. XGB and NN, with MAE of almost 185 minutes on test, perform the best among other algorithms.

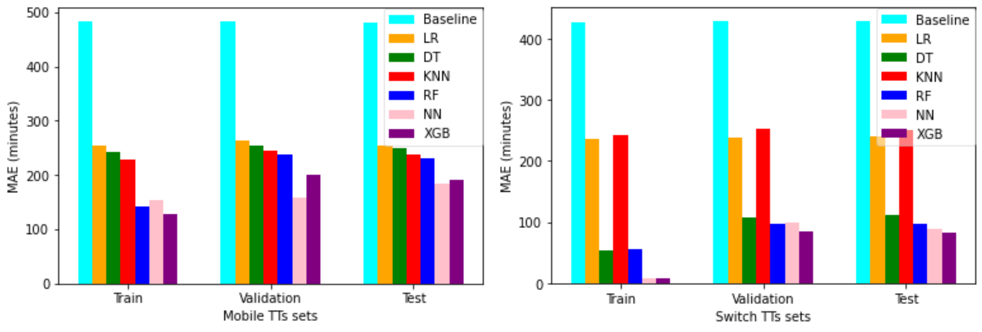


Figure 21 - MAE comparison for resolution time prediction of TTs, (a) Switch, (b) Mobile

Based on the results from switch and mobile TTs, we can draw several conclusions. First, in both systems, there is enough non linearity in the relationship between TT features and the two targets (resolution time and dispatch need) that cannot easily captured by simple models. That is why non-linear models such as NN and XGB show better predictive performance (lower MAE). Second, various characteristics of access switch and radio base station disturbances which are reflected in both data sets results in different predictive performance of the models. Third, we have improved the MAEs compared to company baseline by 80% for switch TTs and 61% for mobile TTs. This is a proof of concept for usefulness of these predictive models in assisting NOC personnel in their daily disturbance handing processes. Tables 15 and 16 show the results of dispatch need prediction on switch and mobile TT test sets at T0. For switch TTs, XGB achieves the best results by predicting 22.39% of the cases that need dispatch ('Yes' label) with a precision of 80.37% and predicting 99.22% of the cases that do not need dispatch ('No' label) with a precision of 89.96% (Recall score indicates the percentage of true positives) and average macro F1 score of 65%. Although LR has the highest recall score of 'No' labels and the lowest FP percentage, it does not perform well in predicting 'Yes' cases, thus acting like a random model, classifying all samples as not needing dispatch. On the other hand, all models can predict 'Yes' cases for the mobile TTs. Among them, again, XGB is the best model by predicting the TTs that need dispatch in 34.46% of the instances and TTs that do not need dispatch in 97.73% of the cases with the precision of, respectively, 65.20% and 92.35% and macro average F1 score of 70%.

Table 15 - percentage of metrics on switch test set

Model	**ND**	**PR**	**RCL**	**F1**	**AUC**	**FP**	**FN**
LR	N	87.51	**1.00**	93.34	63.86	**0.00**	100.00
	Y	0.00	0.00	0.00			
RF	N	88.37	99.18	93.65	68.58	0.82	91.84
	Y	74.53	8.15	14.70			
XGB	N	**89.96**	99.22	**94.36**	**77.65**	0.78	**77.60**
	Y	**80.37**	**22.39**	**35.03**			

where, ND: Need Dispatch, N: No, Y: Yes, PR: Precision, REC: Recall, F1: F1 score, AUC: Area Under the Curve, FP: False Positive (The lower, the better), FN: False Negative (The lower, the better).

Table 16 - Percentage of metrics on mobile test set

Model	ND	PR	RCL	F1	AUC	FP	FN
LR	N	91.15	97.19	94.40	73.92	2.81	76.95
	Y	57.37	23.04	32.88			
RF	N	92.16	97.26	94.64	76.61	2.74	67.01
	Y	59.77	32.98	42.51			
XGB	N	92.35	97.73	94.97	78.46	2.27	65.53
	Y	65.20	34.46	45.09			

We have a low percentage of FP in dispatch need prediction of switch and mobile TTs, indicating that the model has a low false dispatch rate. False workforce dispatch is cost and time-consuming for the company. Therefore, our predictive models perform well in optimizing resource usage. However, we have a high percentage of FN in both cases, showing that the models ignore many cases that need dispatch. We think this is because our data sets are imbalanced. We applied oversampling techniques such as Synthetic Minority Oversampling Technique (SMOTE) [35] to tackle this problem; however, using balancing techniques we observed the trade-off between precision and recall metrics rates, and result could not be improved. Handling imbalanced nature of the problem could be left as a reference point for future works.

Performance w.r.t Increase in Training Samples: We aim to improve the results to provide the most reliable information to NOC and customers. To do so, we investigate if having more data can improve the performance of our predictive models. In other words, will collecting more data over the years result in better predictive performance? To come up with an answer, we train the models on exponentially growing numbers of switch and mobile training samples and evaluate the obtained models on the unchanged sets of test set. Figure 22 shows the MAE convergence of the models. According to this plot, the MAEs are decreasing and flattening to some points, making the changes insignificant. As a result, increasing the number of training samples will not help improve the models' performance. We need to search for other solutions to boost the predictive performance of our models.

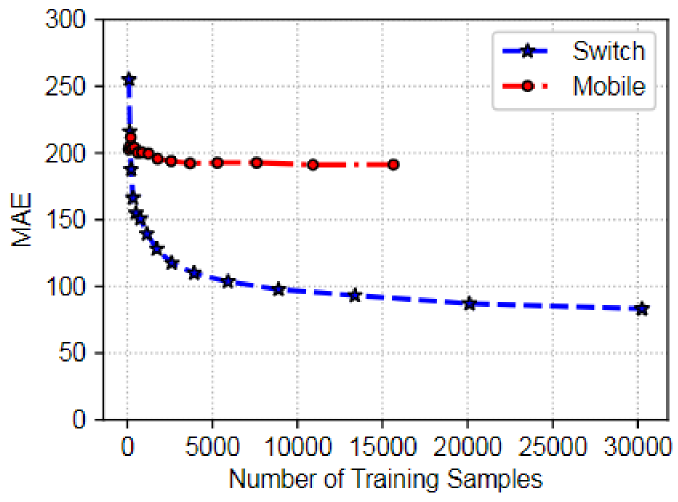
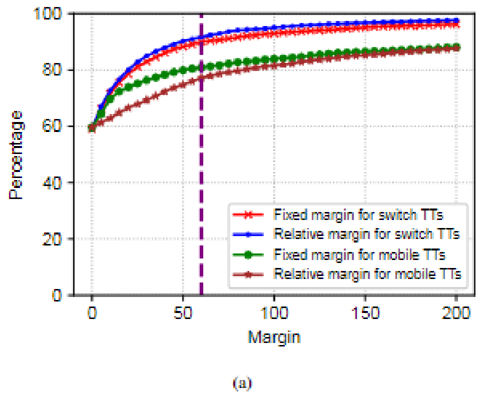


Figure 22 - MAE changes on switch and mobile test sets as training

Resolution time confidence interval for customer satisfaction: Regarding resolution time prediction for the switch and mobile TTs, we devised an approach to propose approximate resolution time ranges within which the TTs will be resolved with a high probability. In other words, to understand the predictive performance of our best regression model (XGB), we investigate what percentages of the test sets are underestimated and overestimated by the models and if marginalizing the predicted values increases the chance of a more reliable prediction. Marginalizing the predicted time refers to increasing it with different values. This can be done in two ways: 1. Adding a fixed value (ex. 25 minutes, 50 minutes, etc.) to the predicted time, or 2. increasing the predicted time by a percentage (ex. 25%, 50%, etc.). Figure 23 (a) show the percentages of switch and mobile TT test samples resolved within their marginalized predicted time. Based on this plot, there is an insignificant difference between resolved percentages of fixed and relative margins for the switch TTs. However, for mobile TTs, more test samples are resolved within the predicted time shifted by fixed margins. According to Figure 23 (b), the MAE changes for mobile TT test samples are almost the same in both fixed and relative marginalization cases; however, for switch TT test samples, the changes in MAE resulted by relatively marginalizing the predicted values are much more severe.

Based on figure 24, we choose 60 minutes as the confidence interval among different fixed margin values. That is because of two reasons: 1. increasing the predicted times by 60 minutes results in the resolution of almost 90% of switch TT test samples and 80% of mobile TT test samples within the estimated ranges (plot a), 2. It also causes the lowest possible increase in MAEs of XGB predictors (plot b). This means reporting a 60-minute marginalized predicted resolution time to the users at the time of disturbance occurrence will guarantee QoE. That is because with a probability of 90% for switch TTs and 80% for mobile TTs, the network disturbance will be resolved within the estimated ranges.



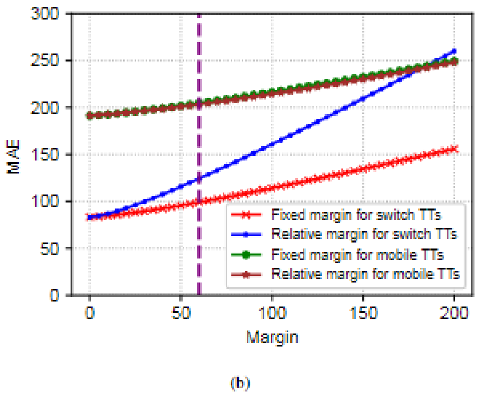
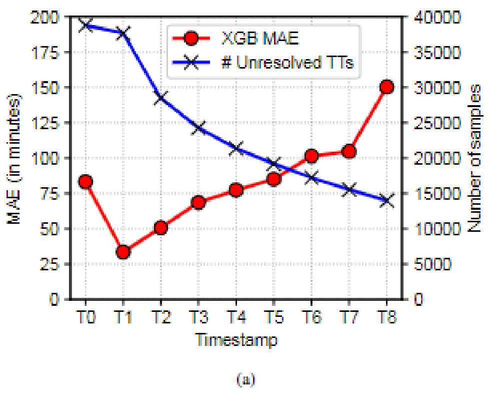


Figure 23 - (a) Percentage of covered switch and mobile TTs' resolution time, (b) MAE changes

Results at Different Timestamps: As mentioned before, the values of some features are updated over time in switch TTs. We would like to understand if this update will result in a better prediction. Thus, we build the models every 15 minutes on the updated set of features. As time passes, more TTs areresolved, specifically, the ones that are resolved automatically. Figure 24 (a) shows the over-time changes in the number of unresolved TTs and MAE of the XGB model for prediction of resolution time. According to this figure, we observe a drop in MAE value at T1. We also have a low portion of test samples resolved at this time (almost 2%). This means because of adding more information to the system within 15 minutes after TT creation, we get 57% improvement in resolution time prediction with the least possible data loss. Figure 24 (b) shows the over-time changes in the number of unresolved TTs and macro average F1 score for dispatch need prediction of switch TTs. In this case, at T1, we have maximum value of the macro F1 metrics. This indicates only after 15 minutes of TT creation; we can predict 95% of TTs that need dispatch and almost 100% of TTs that do not need dispatch with significantly lowest rate of FP and FN.



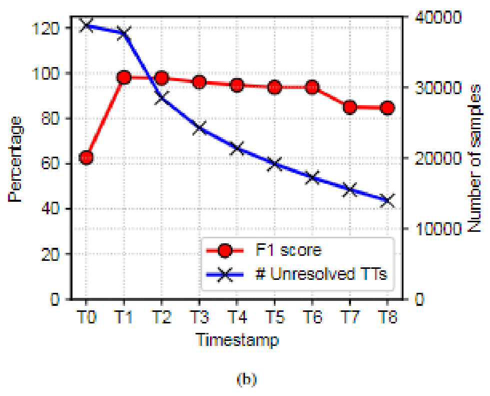


Figure 24 - (a) MAE and unresolved TTs, (b) F1 score and unresolved TTs. For the best model

4.5 Conclusions and Future work

In this work, we presented a proof of concept for an assistant automated system in the telco network disturbance management domain of Telenor Sweden. Our approach was a pragmatic step towards realizing autonomous networks benefiting Telenor customer services delivered through fixed and mobile access networks. We used thousands of historical TTs coming from fixed and mobile network domains. Furthermore, we presented thorough feature engineering and model selection pipelines to tackle the data complexity and heterogeneity. Mainly, we investigated the impact of different TTs' features on two target variables (resolution time and on-site dispatch need), considering their life-cycle evolvement. As far as we know, this is the first time the evolution of network TTs and their impact on target variables has been studied with concrete results. For the first use case, we come up with a 60-minute confidence interval and succeeded to predict the correct resolution time ranges in 90% and 80% of the cases at ticket creation time for respectively, switch and mobile TTs. We had a 80% and 61% improvement over company baseline for this approach. In second case, there is no company baseline, and we achieved an average macro F1 score of 65% and 70% for switch and mobile TTs at creation time. This indicates a significant optimization in workforce (resource)usage of the company.

We also studied the evolution of the switch TTs over time as more information was added to them. We realized that adding more data to TTs within 15 minutes of their creation can improve their resolution time and dispatch need prediction. In the resolution time prediction case, we devised a solution to increase customer satisfaction by choosing a confidence interval. We also provided valuable insights by comparing these two data sets. First, we realized strong non-linearity of data leads to better predictive performance of XGB and NN models. Second, adding more data in both cases did not improve the performance of the models. Third, various characteristics of access switch and radio base station disturbances, reflected in both data sets, result in different predictive performances of the models. To address future works, we consider studying the root causes of network disturbances based on performance data and the impacts on target variables. A thorough research can also be performed to address the imbalanced nature of the data. We can develop approaches for correlating performance data with TTs and draw insights for more autonomous network management.

Following del 4.1 and 4.2 about state-of-the-art approaches and proposed system model for data driven network control and automation towards 10-fold improvement of temporal and spatial allocation of network resources, in Del 4.3, we managed to give an extensive evaluation of the proposed methodology in network disturbance domain and as a result, presented a proof of concept as the support for many future requirements of a self-driven autonomous network. Our approach was a pragmatic step towards realizing autonomous networks benefiting Telenor consumer services delivered through fixed and mobile access network domains. Conclusively, our analysis proved the possibility of meeting SEMANTIC goals by providing an assistant data driven solution for an improved resource optimization.

5 Slice Resource Allocation with Distributed Deep Neural Networks

5.1 Introduction

5G networks are expected to support a large number of tenants simultaneously with different Quality of Service (QoS) requirements and services with different service level agreements (SLA). Satisfying these expectations makes optimal resource allocation a key to the success of 5G networks. However, traditional optimization approaches for this task are often impractical, paving the way for data-driven approaches proposed in recent AI literature [36-39]. Specifically, data-driven algorithms are expected to execute at various network locations, leveraging the computational capabilities at MEC, RAN, and the core in modern cellular networks. Slice resource allocation using deep learning has been a popular recent research direction in the context of 5G networks [40-44]. Also, reinforcement learning has recently been applied to the problem of resource orchestration [41], [43], [45].

Nevertheless, optimization objectives in this context are based on the SLAs and are often asymmetric: i.e., the cost of under-provisioning (penalty paid to the tenant) might differ from the cost of over-provisioning (opportunity cost of wasted resources). However, running a centralized heavy duty DNN faces two key challenges in (5G+) wireless architectures: (i) a number of network optimization tasks, especially those at the RAN, have stringent latencies compared to UE-level applications often offloaded to the cloud. (ii) the overhead of sending raw data over potentially congested edge/wireless links can often be a major hurdle in the application of such solutions.

Motivated by the research done in [46-51], in this work, we attempt to further show the generality of distributed deep neural networks methodology by investigating how to distribute a more sophisticated DNN architecture for the same task, based on LSTM (Long Short Term Memory) units. Specifically, the main contributions of this work are the following:

- We propose a distributed LSTM architecture for the problem of balancing under-/over-provisioning of resources to different slices and investigate how the methodology of DDNNs can be applied to such larger, more sophisticated architectures (compared to that of [46]).
- We demonstrate the impact of properly tuning the joint training hyperparameters of local and remote "exits" (i.e., predicted allocations and related SLA costs) to achieve a good balance between: (i) making the local layers powerful enough to correctly make a large enough number of allocation decisions, while (ii) producing useful features that the remote layers could leverage, when improved allocation decisions are deemed necessary.
- We propose a mechanism that measures the confidence in the local exit, and predicts whether the remote exit (that requires additional latency and communication) would improve the SLA costs enough to justify the extra overhead (i.e., a type of unsupervised learning).

5.2 Problem setup

Assume we have a set of K network functions (e.g. VNFs) or network elements (e.g. BSs). Each network function/element requires some resources which will be allocated according to its traffic demand (e.g. each base station requires some Band Width (BW)). We can consider the DNN as a black box and model it with an approximation function with some parameters that takes an input vector and gives the predicted value. Therefore we can write:

$$\hat{y}_t^i = DNN\left(\boldsymbol{d}_{t,N}^i, \boldsymbol{\theta}\right) \qquad (5)$$

Where $\boldsymbol{d}_{t,N}^i$ is the input vector for the DNN. The input vector $\boldsymbol{d}_{t,N}^i = \{d_{t-N}^i, \dots, d_{t-1}^i\}$ consists of the N past traffic samples of BS $i \in K$ before the time t. N is the input vector size and is constant during the model training. $DNN(:, \boldsymbol{\theta})$ is the approximation function with $\boldsymbol{\theta}$ as parameters. The vector $\boldsymbol{\theta}$ is the model parameters (i.e. weights of the DNN). $\hat{y}_t^i$ is the predicted traffic demand for BS i at time t. DNN will be trained to predict the best approximation of the real traffic demand for BS i at time t, i.e. d_t^i. The objective function determines how good the prediction value is.

In a standard forecasting problem, one wants to predict the traffic value at time t using the past N traffic samples $\boldsymbol{d}_{t,N}^i = \{d_{t-N}^i, \dots, d_{t-1}^i\}$. The goal is that the predicted value $\hat{y}_t$ be as close as possible to the real traffic d_t. To achieve this we can train a DNN with a least squares objective function.

$$f(\hat{y}_t, d_t) = (\hat{y}_t - d_t)^2 \qquad (6)$$

As mentioned before, the goal is to predict a value that is as close as possible to the real value and the fact that the predicted value can be higher or lower the real value doesn't matter. A key difference in our work is that, the goal is not just to predict a value, the predicted traffic demand will be used to allocate resources to network elements. In this case, predicted traffic being less or more than the real traffic is very important. If the predicted traffic is less than the needed traffic, then not enough resources will be allocated to the corresponding network element; this is called underprovisioning ($\hat{y}_t < d_t$) and could violate the Service Level Agreement (SLA) with the slice tenants. If the predicted traffic is more than the needed traffic, then more resources than what is needed will be allocated to the corresponding network element; this is called overprovisioning ($\hat{y}_t > d_t$) and although tenants are satisfied but some resources will not be used.

To this end, we are looking for an objective function that makes the DNN to avoid underprovisioning and to minimize overprovisioning. One recommended objective function is:

$$f(\hat{y}_t, d_t) = \begin{cases} c - \epsilon(\hat{y}_t - d_t) & if\ (\hat{y}_t - d_t) \leq 0 \\ c - \frac{1}{\epsilon}(\hat{y}_t - d_t) & if\ 0 < (\hat{y}_t - d_t) \leq \epsilon c \\ (\hat{y}_t - d_t) - \epsilon c & if\ (\hat{y}_t - d_t) > \epsilon c \end{cases} \quad (7)$$

where ϵ is a very low constant value that is used just to help the stochastic gradient descent (SGD) in DNN operate correctly. Constant c is a penalty that tries to avoid underprovisioning and the linear penalty tries to minimize overprovisioning. This function is shown in Figure 25.

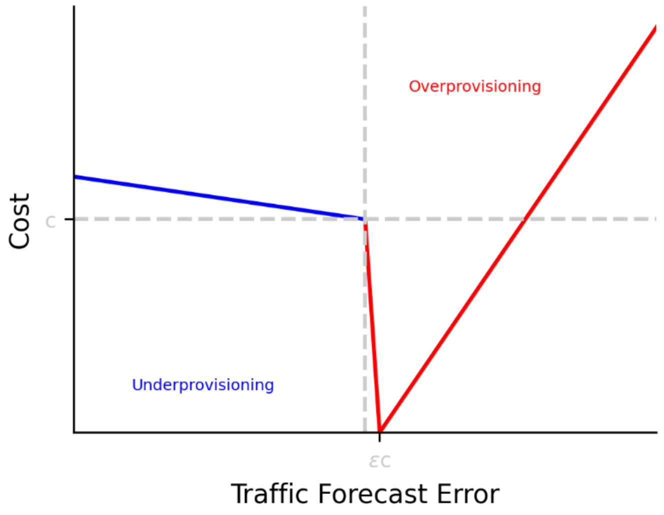


Figure 25 - Cost function with c=0.5,ϵ=0.1

Another objective function that can be used is:

$$f(\hat{y}_t, d_t) = \begin{cases} c + c_1(\hat{y}_t - d_t)^2 & if\ (\hat{y}_t - d_t) \leq 0 \\ c - \frac{1}{\epsilon}(\hat{y}_t - d_t) & if\ 0 < (\hat{y}_t - d_t) \leq \epsilon c \\ (\hat{y}_t - d_t) - \epsilon c & if\ (\hat{y}_t - d_t) > \epsilon c \end{cases} \quad (8)$$

This objective function suggests to penalize underprovisioning quadratically. The function is shown in figure 26.

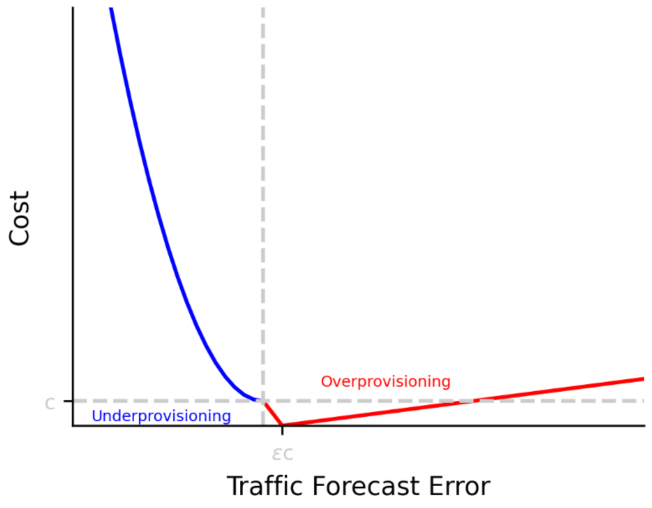


Figure 26 - Cost function with c=0.5, c_1=50, ϵ=0.1

Without loss of generality, we'll assume the following objective:

$$f(\hat{y}_t, d_t) = \begin{cases} c_1(\hat{y}_t - d_t)^2 & if\ (\hat{y}_t - d_t) \leq 0 \\ c_2(\hat{y}_t - d_t) & if\ (\hat{y}_t - d_t) > 0 \end{cases} \qquad (9)$$

Where SLA violations has quadratic penalty and overprovisioning penalty is linear. This function is illustrated in figure 27.

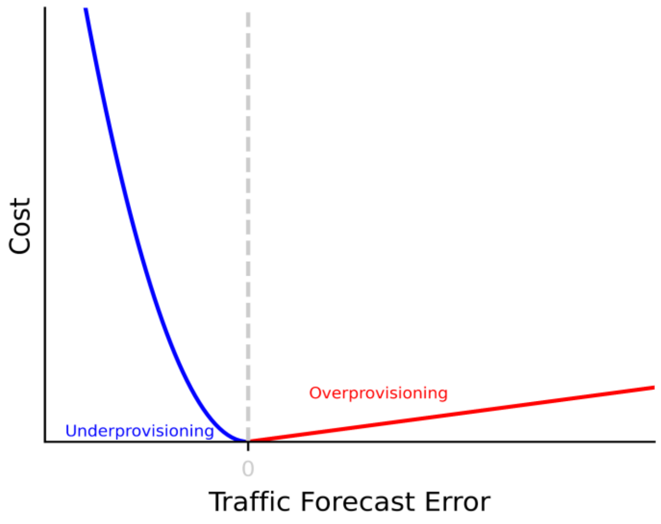


Figure 27 - Cost function with c_1=50, c_2=1

5.3 System model

In this section, we propose a Distributed Deep Neural Network (DDNN) architecture over a 5G network setup. The DDNN runs a small subset of DNN layers at edge (e.g. RAN) and more layers in cloud (e.g. MEC).

We assume a 5G network in which a set of BSs require some resources. Each BS at time t demands an amount of resources (e.g. BW), d_t^i, to fulfill its corresponding users SLAs. We have the past N demand values $\boldsymbol{d}_{t,N}^i = \{d_{t-N}^i, \ldots, d_{t-1}^i\}$. Traffic demand samples are random and possibly non-stationary. The vector $\boldsymbol{d}_{t,N}^i$ is given to the DDNN to predict the demand for each BS at time t, $\hat{y}_t^i$. We can describe the DDNN with the following equation:

$$\left(\hat{y}_{L,t}^i, \hat{y}_{R,t}^i\right) = DDNN\left(\boldsymbol{d}_{t,N}^i; \boldsymbol{\theta}\right) \qquad (10)$$

Where the $DDNN(;\boldsymbol{\theta})$ is the approximation function that models the DDNN and $\boldsymbol{\theta}$ is the model parameters. $\hat{y}_{L,t}^i$ and $\hat{y}_{R,t}^i$ are the output of the local and remote exits respectively. We can consider the DDNN as two DNN that are connected to each other as shown in figure 28. One DNN in the local and another in the remote.

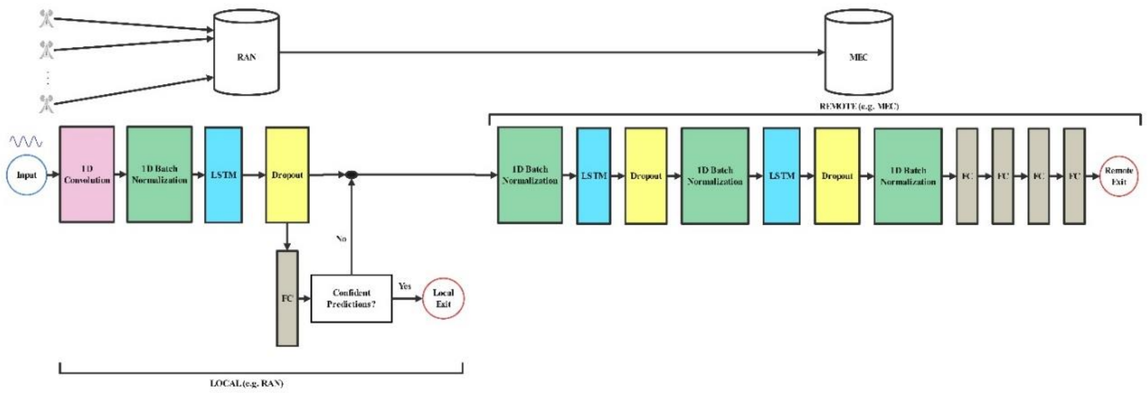


Figure 28 - LSTM network is distributed over RAN and MEC

As shown in figure 28, we use a Three Layer LSTM network for the prediction. The deep neural network is distributed over RAN which is the local part and MEC which is the remote part.

Local Exit: At the local DNN, first we put a 1 dimensional convolution. The 1D convolution block is used as feature detector for time series input data. Then, there is a 1 dimensional batch normalization which accelerates deep network training by reducing internal covariate shift. We use the default batchnorm1d pytorch parameters for this block. After that, we put 1 layer LSTM block. This LSTM layer has just 1 unit, comparing to the number of units in the next layers, the LSTM in local is very simple. Then, there is a dropout layer for regularization. The output will be sent to a Fully Connected (FC) layer and also will be stored and in case needed will be sent to remote layers. The FC layer in local is linear, the output of this layer is called ***local predictions***, i.e. $\hat{y}^{i}_{L,t}$.

In order to have fast resource allocation and also lower computation cost, we are looking for high number of locally resolved samples. If the local prediction is "good enough" then it will be the traffic demand prediction otherwise the data will be sent to remote where there are additional NN layers to forecast the traffic demand. The mechanism gives a confidence signal, i.e. YES/NO about local predictions. The output of FC layer in local is given to the confidence mechanism and it decides whether the DNN in local is confident about the prediction which means the local prediction is good or the sample need to be processed more which means data should be sent to remote layers.

Remote Exit: The output of the local dropout layer will be sent to the remote. At the remote DNN, first there is a 1 dimensional batch normalization with default parameters. Then, there is a LSTM layer with 128 units. After that, there is a dropout layer followed by a 1 dimensional batch normalization with default parameters. Then, there is the last LSTM layer with 64 units again followed by a dropout layer and a 1 dimensional batch normalization with default parameters. Then, there are four FC layers. The first FC has 128 hidden neurons with ReLu activation function. The second FC has 64 hidden neurons with ReLu activation function. The third FC has 32 hidden neurons (linear layer), and the last FC layer make the predictions. The output of the last FC layer in remote DNN is called ***remote predictions***, i.e. $\hat{y}^{i}_{R,t}$. If the local prediction was not good enough, decided by confidence mechanism, then the traffic demand prediction is the remote prediction.

5.4 Model Training

While training a centralized DNN is quite straightforward, DDNN training requires **jointly training** the local and remote DNN modules towards achieving a common goal. Joint training means weighing both the local and remote exits in the objectives as follows and allowing the error of both to backpropagate through their respective DNN layers. The DDNN overall loss is calculated as follows:

$$Loss_{DDNN} = \sum_{m=1}^{M} w_L * f(\hat{y}_{L,m}, d_m) + w_R * f(\hat{y}_{R,m}, d_m) \qquad (11)$$

In Equation (11), w_L is the "local weight" and w_R is the "remote weight", which regulate the impact of the local and remote exit on the overall loss of the DDNN during the joint training. These weights, $w_L, w_R \in [0,1]$ and $w_R = 1 - w_L$, play an important role in the optimization process. If we set $w_L = 0$ (which also means $w_R = 1$) then the DDNN resembles a centralized DNN and tries to optimize the performance in the remote exit. Similarly, if we set $w_L = 1$ (which also means $w\_R = 0$) then the DDNN tries to optimize the performance in the local exit. Selecting the best (w_L, w_R) is crucial in order to achieve the desired goals (i.e., good local predictions, allocating resources locally for many slices, and good remote predictions).

Oracle-based Offloading: First, let us assume an "oracle" that know the potential added value of remote processing for any sample. We will use this as reference. We can calculate the loss difference:

$$LD = f(\hat{y}_{L,m}, d_m) - f(\hat{y}_{R,m}, d_m) \qquad (12)$$

If, for example, we want to offload for example 40% of the samples locally, then we pick 40% of the samples with the lowest LD for offloading in the local exit and the remaining samples, which have higher LD, will be exited remotely. Hence, if we did have such an oracle, we could certainly always keep the local decisions that will be better than the remote ones. While among samples for which the remote exit is better, we would choose to keep the local ones that are closest to the remote ones. Finally, when Equation (12) is large, this suggests that the remote exit could offer significant cost benefits (hence justifying the additional overhead).

Bayesian Confidence-based Offloading: Unfortunately, in practice we do not have such an oracle, as the right term of Equation (12), i.e., the remote decision $\hat{y}_{R,m}$ and the related cost, cannot be known at the edge, without actually sending the sample to the remote cloud. We propose a methodology based on random dropouts, applied to the local forward pass, motivated by the Bayesian confidence metric in [47] (this dropout is different from the dropout block used as a regularizer during the training)

The confidence block includes a dropout layer with dropout probability $p = 0.4$, followed by a linear FC block. The intermediate signal z is given to the confidence block and it is forced to infer for each input sample, say $J = 10$ times. The randomness of the dropout makes the inference of the confidence block different at each time. Therefore, for each base station, we have an array $\in R^J$. For each base station. $k \in K$, we calculate the standard deviation (σ_k) and then take the average among the K base stations. We refer to this value as Uncertainty:

$$U = \frac{1}{k} \sum_{k=1}^{K} \sigma_k \qquad (13)$$

This metric serves as a worst case estimate of how much perturbations have affected the local decisions. The confidence mechanism compares the measured uncertainty (U) value with a given confidence threshold (η), which is a design parameter of the DDNN. If $U < \eta$, then the model is confident about the local decision, and it is considered "good enough". Otherwise, the intermediate signal z will be sent to the remote layers, where the remote decision is assumed to be the correct decision.

5.5 Performance Evaluation

Resource Allocation and Communication Trade-off: After jointly training the model, for each η (confidence threshold) in $[0, 1]$ we measure the samples in the test set that can be exited locally and then calculate the total loss, as in Equation (11). We plot the trade-off curve, which represents the total loss versus the percentage of samples resolved in the local exit. When we use Oracle-based Offloading, we obtain the offline trade-off curve. By using Bayesian Confidence-based Offloading, we produce the online trade-off curve.

In Figure 29 we plot both offline and online trade-off curves for the model with (local, remote) training weights $(w_L, w_R) = (0.9, 0.1)$. We can see that the Bayesian offloading mechanism (blue curve) has a good performance compared to the oracle offloading (black curve), which represents the ideal offloading policy. We can see that when 40% (or less) of the samples are exited locally, the oracle and Bayesian mechanisms have similar performance.

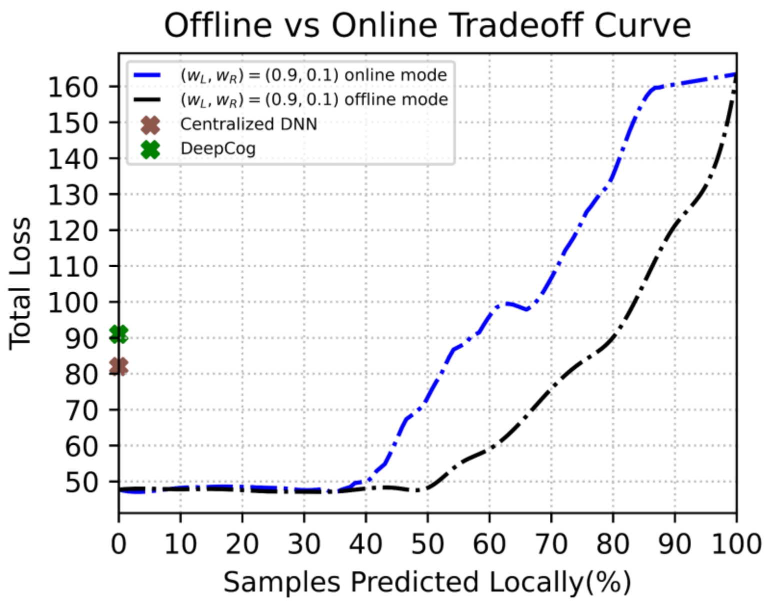


Figure 29 - Total loss vs percentage of samples exited locally with $(w_L, w_R) = (0.9{,}0.1)$

The online trade-off curve for three models is shown in figure 30. We repeat the process for three DDNN models each using the following (local, remote) training weights, respectively: (0.8, 0.2), (0.87, 0.13) and (0.9, 0.1). In this figure, we also mark the centralized DNN loss and the DeepCog loss. The centralized LSTM architecture indeed slightly outperforms the centralized CNN-based architecture of DeepCog, as expected. (We remind you that our main goal here is not to improve the DNN architecture itself, but rather to investigate how to distribute more complex, memory-based DNNs for such tasks).

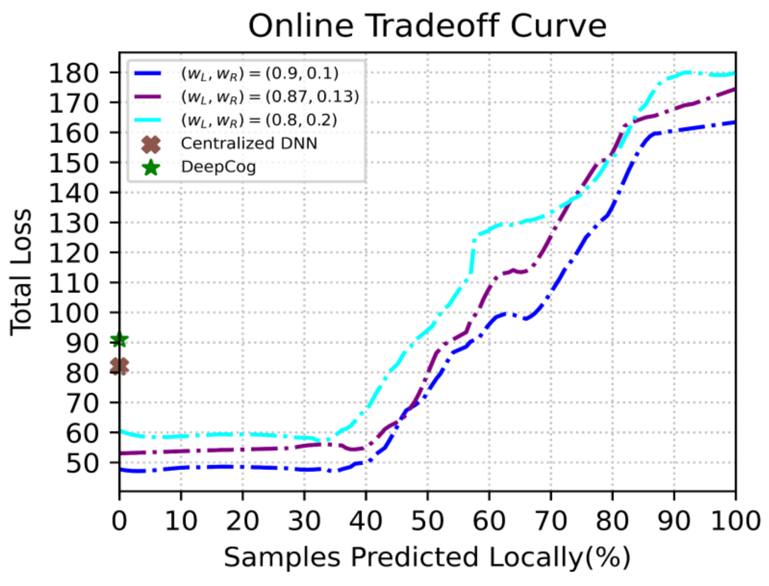


Figure 30 - Total loss vs percentage of samples exited locally for three models

As we can see in figure 30 the $(w\_L, w\_R) = (0.9, 0.1)$ model achieves the best trade-off in the considered scenario. It is possible to resolve more than 50% of the samples locally while the overall loss equals that of the centralized DNN. The $(w\_L, w\_R) = (0.8, 0.2)$ model, can resolve all the samples remotely, i.e., $\backslash eta = 0$, while the total loss is nearly 25% less than that of the centralized DNN loss. Also, with this model, we can offload more than 40% of the samples locally while the cost is the same as that of the centralized DNN.

It is evident that the overall trade-off curve is affected by the choice of the (offline) training weights. In figure 31, the online trade-off curve for the model with $(w\_L, w\_R) = (0.1, 0.9)$ is shown (by increasing the confidence threshold η, we can plot it). We observe that with this pair of weights, the model doesn't perform well and in fact, its loss is always more than that of the centralized models. We have also analyzed other weight combinations, yet it is clear that even for "non-optimal" weight choices, there is still an interesting trade-off achieved. Another important observation is that, in our model a local weight higher than the remote weight ($w_L > w_R$ or $w_L > 0.5$) is needed to counterbalance the fact that the local module is much simpler/shallower, and be able to surpass the centralized DNN performance.

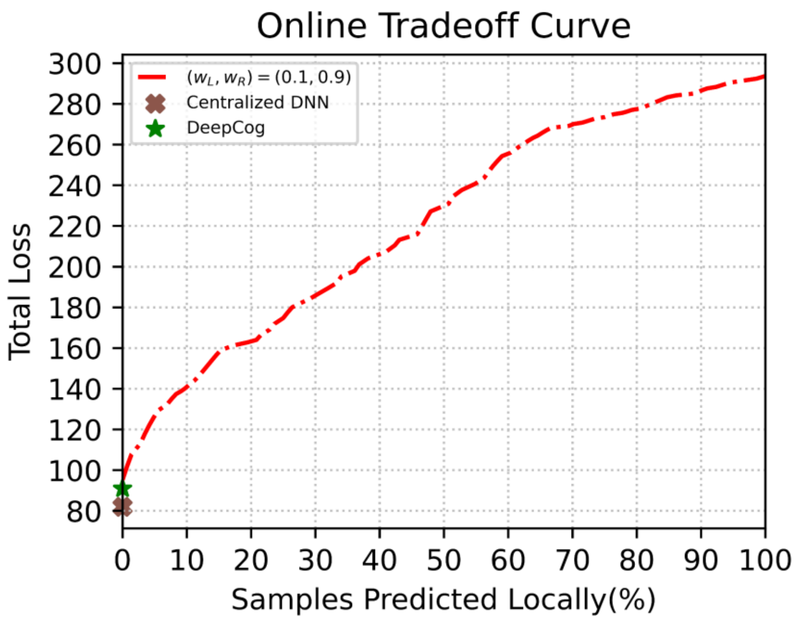


Figure 31 - Total loss vs percentage of samples exited locally with $(w_L, w_R) = (0.1, 0.9)$

SLA Violations Avoidance: In Figures 32 and 33, the real traffic demand (BandWidth) (d) in the test set, the local allocations ($\hat{y}_L$), and the remote allocations ($\hat{y}_R$) for one of the base stations with two different (local, remote) training weights $(w_L, w_R) = (0.9, 0.1)$\$ and $(w_L, w_R) = (0.1, 0.9)$ have been illustrated. To obtain local and remote predictions for all samples, we don't use the confidence mechanism. In figure 32, we can see that with $(w_L, w_R) = (0.9, 0.1)$ the local exit has good performance and some samples can be predicted locally, while figure 33 shows that with $(w_L, w_R) = (0.1, 0.9)$, the local exit doesn't work well and using local predictions increases the cost which was expected according to the trade-off curve for this weight pair in Figure 31. Also, we observe that both models avoid SLA violations, rather than trying to "match" the demand (as an MSE objective would), due to the higher cost of under-provisioning in our objective function.

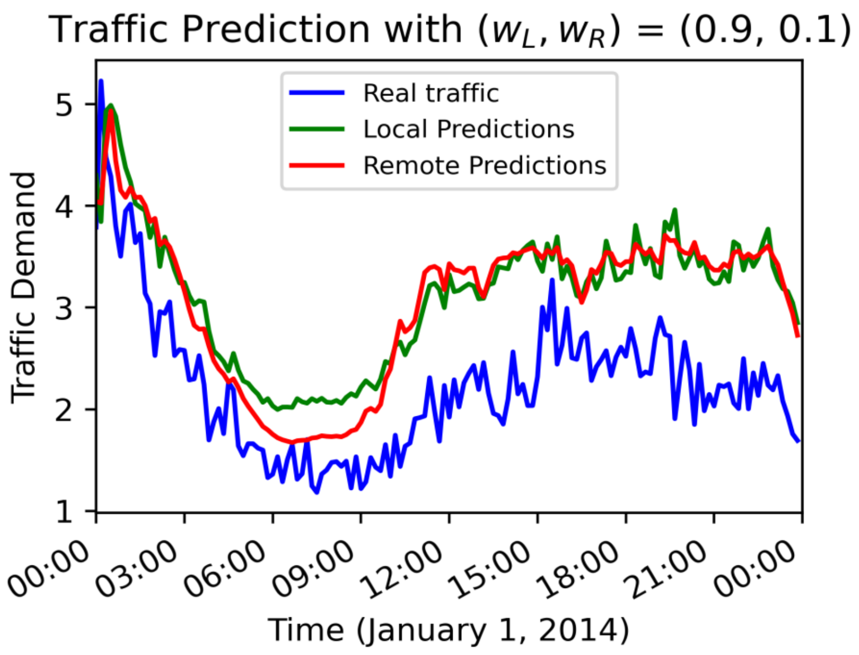


Figure 32 - Real traffic demand with $(w_L, w_R) = (0.9,0.1)$

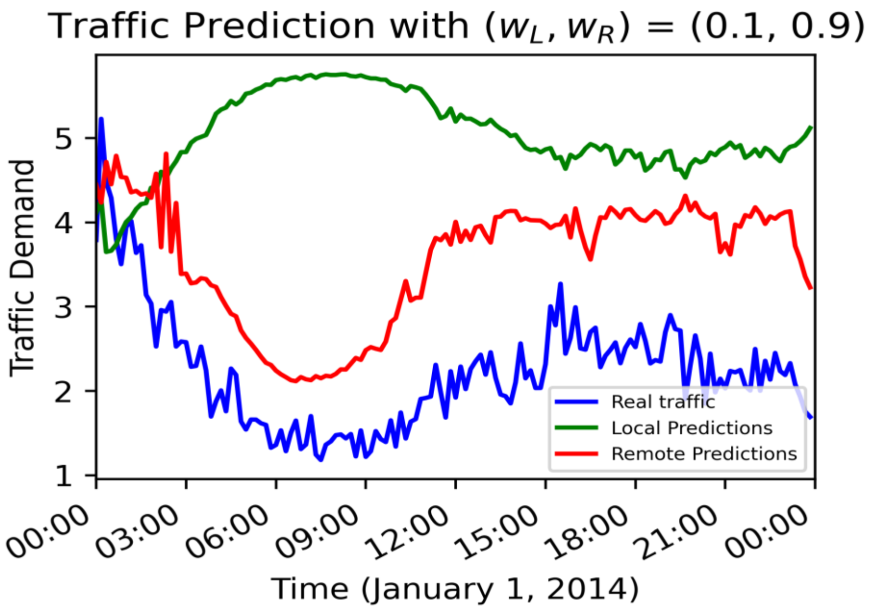


Figure 33 - Real traffic demand with $(w_L, w_R) = (0.1,0.9)$

5.6 Conclusions and Future work

We designed and implemented a Distributed Deep Neural Network (DDNN) for forecasting future traffic demand and allocating resources accordingly in 5G+ networks. The proposed DDNN is a DNN with multiple exit points: one local exit (e.g., Edge) and one remote exit (e.g., Cloud). The DDNN needs to be trained jointly to achieve the desired goals. During joint training, a weight is assigned to the local exit, and another weight is assigned to the remote exit, which encourages good performance at the local exit and also affects the performance of the remote exit. Additionally, the objective function plays an important role in avoiding under-provisioning. We use a Bayesian confidence mechanism to determine either the samples should offload locally or remotely. In comparison with centralized models, our algorithm can achieve lower or equal cost performance while resolving more than 50% of the samples locally and also reducing latency.

We are currently investigating alternative confidence mechanisms to reduce computational usage and accelerate the allocation process in online mode. Additionally, we are considering implementing the model with multiple local exits. As the local and remote weights play an important role, adaptive tuning of local and remote weights (w_L, w_R) during training could be an interesting topic for future work.

6 Conclusions

This document acts as a summary of the efforts that have been done towards SEMANTIC Work Package 4. After a short introduction, deliverable D4.3 presented a description of the problems currently being tackled by each ESR, their analysis, approach, proposed solution, and the simulation results and analytical explanations.

List of Acronyms and Abbreviations

Acronym	***Description***
3D-DefCNN	3D Deformable Convolutional Neural Networks
5G	5th generation (5G) mobile network
6G	6th generation (6G) mobile network
AF	Application Function
AI/ML	Artificial Intelligence / Machine Learning
AP	Access Point
BBU	Baseband Unit
BS	Base Station
BW	Bandwidth
CMDP	Constrained Markov Decision Process
CNN	Convolutional Neural Network
CRAN	Cloud Radio Access Network
CQI	Channel Quality Indicator
DDNN	Distributed Deep Neural Network
DNN	Deep Neural Network
E2ENS	End-to-End Network Slicing
EMBB/eMBB	Enhanced Mobile Broadband
FC	Fully Connected
FNN	Feedforward Neural Network
IPO	Interior-point Policy Optimization
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MEC	Multi-Access Edge Computing
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
mMTC	massive Machine Type Communication
MNO	Mobile Network Operator
MTD	Mobile Traffic Decomposition
NOC	Network Operation Center
NN	Neural Network
O-RAN	Open-RAN
OSS	Operations Support System
PPP	Poisson Point Process
QoE	Quality of Experience
QOS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
ReLU	Rectified Linear Unit
RIC	RAN Intelligence Controller
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RRH	Remote Radio Head
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSSI	Reference Signal Strength Indicator
RT	Resolution Time
SGD	Stochastic Gradient Descent
SINR	Signal to Interference Noise Ratio

SLA	Service Level Agreement
TT	Trouble Ticket
UE	User Equipment
URLLC	Ultra-Reliable Low Latency Communication
VNF	Virtual Network Function

References

[1] Z. Zhang et al., “6g wireless networks: Vision, requirements, architecture, and key technologies,” IEEE Veh. Technol. Mag., vol. 14, 2019.

[2] T. Rashid et al., “QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning,” in 35th ICML, 2018. [3] R. Chataut and R. Akl, “Massive mimo systems for 5g and beyond networks—overview, recent trends, challenges, and future research direction,” Sensors, vol. 20, no. 10, 2020.

[3] J. Yang et al., “Hierarchical cooperative multi-agent reinforcement learning with skill discovery,” 2019.

[4] P. Sunehag et al., “Value-decomposition networks for cooperative multiagent learning,” arXiv preprint arXiv:1706.05296, 2017.

[5] J. Mei et al., “Intelligent radio access network slicing for service provisioning in 6g: A hierarchical deep reinforcement learning approach,” IEEE Trans. Commun., vol. 69, no. 9, 2021.

[6] R. Li et al., “Deep reinforcement learning for resource management in network slicing,” IEEE Access, vol. 6, 2018.

[7] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” in International conference on machine learning. PMLR, 2016, pp. 2829–2838.

[8] E. U. T. R. Access, “Self-configuring and self-optimizing network use cases and solutions,” Protocol specification (Release 9), 2011.

[9] L. Chen, D. Yang, D. Zhang, C. Wang, J. Li and others, “Deep mobile traffic forecast and complementary base station clustering for C-RAN optimization,” Journal of Network and Computer Applications, vol. 121, p. 59–69, 2018.

[10] O. Østerbø and O. Grøndalen, “Benefits of self-organizing networks (SON) for mobile operators,” Journal of Computer Networks and Communications, vol. 2012, 2012.

[11] R. N. Gemaque, A. F. J. Costa, R. Giusti and E. M. Dos Santos, “An overview of unsupervised drift detection methods,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 10, p. e1381, 2020.

[12] G. E. Hinton, T. J. Sejnowski and others, “Learning and relearning in Boltzmann machines,” Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1, p. 2, 1986.

[13] S. Paindi Jayakumar and A. Conte, “Framework: Clustering-Driven Approach for Base Station Parameter Optimization and Automation (CeDA-BatOp),” 2024.

[14] A. Benton, H. Khayrallah, B. Gujral, D. A. Reisinger, S. Zhang and R. Arora, “Deep generalized canonical correlation analysis,” arXiv preprint arXiv:1702.02519, 2017.

[15] G. Andrew, R. Arora, J. Bilmes and K. Livescu, “Deep canonical correlation analysis,” in International conference on machine learning, 2013.

[16] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” science, vol. 313, p. 504–507, 2006.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, p. 8024–8035.

[18] O. R. A. N. Community, Ric Applications (RICAPP), 2022.

[19] M. Liyanage, Q.-V. Pham, K. Dev, S. Bhattacharya, P. K. R. Maddikunta,T. R. Gadekallu, and G. Yenduri, “A survey on zero touch network and

[20] J. Rexford, “Evolving toward a self-managing network.”

[21] W. Sun and A. Isac, “How to automatically resolve trouble tickets with machine learning,” 2020. [Online]. Available: https://www.ericsson.com/

[22] A. Björling, “Telecommunications trouble ticket resolution time modelling with machine learning,” 2021.

[23] R. Colella, “Trouble tickets resolution time estimation: The design of a solution for a real case scenario,” 2021.

[24] N. L. Crookston and A. O. Finley, “yaimpute: an r package for knn imputation,” Journal of Statistical Software, vol. 23, pp. 1–16, 2008.

[25] D. A. Freedman, Statistical models: theory and practice. Cambridge university press, 2009.

[26] L. E. Peterson, “K-nearest neighbor,” Scholarpedia, vol. 4, no. 2, p.1883, 2009.

[27] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and regression trees. Routledge, 2017.

[28] L. Breiman, “Random forests,” Machine learning, vol. 45, no. 1, pp.5–32, 2001.

[29] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,”in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016, pp. 785–794.

[30] D. J. MacKay, “Bayesian interpolation,” Neural computation, vol. 4, no. 3, pp. 415–447, 1992.

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

[32] K. P. Murphy, Machine learning: a probabilistic perspective. MIT press, 2012.

[33] G. James, D. Witten, T. Hastie, R. Tibshirani et al., An introduction to statistical learning. Springer, 2013, vol. 112.

[34] J. Heaton, “Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618,” Genetic programming and evolvable machines, vol. 19, no. 1-2, pp. 305–307, 2018.

[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” Journal of artificial intelligence research, vol. 16, pp. 321–357, 2002.

[36] C. Zhang, P. Patras and H. Haddadi, “Deep Learning in Mobile and Wireless Networking: A Survey,” in IEEE Communications Surveys and Tutorials, vol. 21, no. 3, pp. 2224-2287, 2019.

[37] C. -X. Wang, M. D. Renzo, S. Stanczak, S. Wang and E. G. Larsson, “Artificial Intelligence Enabled Wireless Networking for 5G and Beyond: Recent Advances and Future Challenges,” in IEEE Wireless Communications, vol. 27, no. 1, pp. 16-23, February 2020.

[38] D. P. Kumar, T. Amgoth, C. S. R. Annavarapu, “Machine learning algorithms for wireless sensor networks: A survey,” Information Fusion, vol. 49, pp. 1-25, ISSN 1566-2535, 2019.

[39] M. Chen, U. Challita, W. Saad, C. Yin and M. Debbah, "Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial," in IEEE Communications Surveys and Tutorials, vol. 21, no. 4, pp. 3039-3071, 2019.

[40] H. Halabian, "Distributed Resource Allocation Optimization in 5G Virtualized Networks," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 3, pp. 627-642, March 2019.

[41] Y. Liu, J. Ding and X. Liu, "A Constrained Reinforcement Learning Based Approach for Network Slicing," 2020 IEEE 28th International Conference on Network Protocols (ICNP), Madrid, Spain, pp. 1-6, 2020.

[42] C. Zhang, M. Fiore, C. Ziemlicki, P. Patras, "Microscope: Mobile Service Traffic Decomposition for Network Slicing as a Service," MobiCom '20: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, no. 38, pp. 1–14, April 2020.

[43] V. Sciancalepore, X. Costa-Perez and A. Banchs, "RL-NSB: Reinforcement Learning-Based 5G Network Slice Broker," in IEEE/ACM Transactions on Networking, vol. 27, no. 4, pp. 1543-1557, August 2019.

[44] J. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore and X. Costa-Perez, "Overbooking Network Slices through Yield-Driven End-to-End Orchestration," CoNEXT '18: Proceedings of the 14th International Conference on emerging Networking Experiments and Technologies, pp. 353–365, December 2018.

[45] Q. Liu, T. Han, and E. Moges, "Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning," IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, Singapore, pp. 234-244, 2020.

[46] D. Bega, M. Gramaglia, M. Fiore, A. Banchs and X. Costa-Perez, "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning," IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, Paris, France, pp. 280-288, 2019.

[47] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Aztec: Anticipatory capacity allocation for zero-touch network slicing," IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, Toronto, ON, Canada, pp. 794-803, 2020.

[48] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, pp. 328-339, 2017.

[49] O. Nassef, W. Sun, H. Purmehdi, M. Tatipamula, and T. Mahmoodi, "A survey: Distributed Machine Learning for 5G and beyond," Computer Networks, vol. 207, ISSN 1389-1286, 2022.

[50] X. Chen, C. Wu, Z. Liu, N. Zhang and Y. Ji, "Computation Offloading in Beyond 5G Networks: A Distributed Learning Framework and Applications," IEEE Wireless Communications, vol. 28, no. 2, pp. 56-62, April 2021.

[51] T. Giannakas, T. Spyropoulos and O. Smid, "Fast and accurate edge resource scaling for 5G/6G networks with distributed deep neural networks," 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Belfast, United Kingdom, pp. 100-109, 2020.