

Article

Redefining 6G Network Slicing: AI-Driven Solutions for Future Use Cases

Robert Botez , Daniel Zinca  and Virgil Dobrota * 

Communications Department, Technical University of Cluj-Napoca, 400114 Cluj-Napoca, Romania; robert.botez@com.utcluj.ro (R.B.); daniel.zinca@com.utcluj.ro (D.Z.)

* Correspondence: virgil.dobrota@com.utcluj.ro

Abstract: The evolution from 5G to 6G networks is driven by the need to meet the stringent requirements, i.e., ultra-reliable, low-latency, and high-throughput communication. The new services are called Further-Enhanced Mobile Broadband (feMBB), Extremely Reliable and Low-Latency Communications (ERLLCs), Ultra-Massive Machine-Type Communications (umMTCs), Massive Ultra-Reliable Low-Latency Communications (mURLLCs), and Mobile Broadband Reliable Low-Latency Communications (MBRLLCs). Network slicing emerges as a critical enabler in 6G, providing virtualized, end-to-end network segments tailored to diverse application needs. Despite its significance, existing datasets for slice selection are limited to 5G or LTE-A contexts, lacking relevance to the enhanced requirements. In this work, we present a novel synthetic dataset tailored to 6G network slicing. By analyzing the emerging service requirements, we generated traffic parameters, including latency, packet loss, jitter, and transfer rates. Machine Learning (ML) models like Random Forest (RF), Decision Tree (DT), XGBoost, Support Vector Machine (SVM), and Feedforward Neural Network (FNN) were trained on this dataset, achieving over 99% accuracy in both slice classification and handover prediction. Our results highlight the potential of this dataset as a valuable tool for developing AI-assisted 6G network slicing mechanisms. While still in its early stages, the dataset lays a foundation for future research. As the 6G standardization advances, we aim to refine the dataset and models, ultimately enabling real-time, intelligent slicing solutions for next-generation networks.

Keywords: 6G; deep learning; ERLLC; feMBB; IoT; MBRLLC; machine learning; mURLLC; network slicing; umMTC



Academic Editor: Djuradj Budimir

Received: 24 December 2024

Revised: 10 January 2025

Accepted: 16 January 2025

Published: 18 January 2025

Citation: Botez, R.; Zinca, D.; Dobrota, V. Redefining 6G Network Slicing: AI-Driven Solutions for Future Use Cases. *Electronics* **2025**, *14*, 368. <https://doi.org/10.3390/electronics14020368>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The rapid proliferation of mobile devices and the rise of innovative applications have driven advancements in wireless network technologies. Fifth-generation (5G) networks, with their core services—enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communication (URLLC), and massive Machine-Type Communication (mMTC)—have laid the foundation for high-speed connectivity and diverse use cases. These services have enabled breakthroughs in areas such as industrial automation, remote healthcare, and connected vehicles. However, the rapid proliferation of the Internet of Things (IoT) and the increasing complexity of future applications reveal limitations in the capabilities of 5G to fully address the demands of tomorrow’s hyper-connected ecosystem [1–3].

The upcoming sixth generation (6G) networks aim to bridge these gaps by redefining wireless communication paradigms. Unlike 5G, which primarily extends LTE technologies, 6G networks envision a data-driven, AI-native framework that integrates communication, computing, and sensing to meet the stringent demands of emerging applications

such as extended reality (XR), brain–computer interfaces, and connected autonomous systems. These applications need ultra-high reliability, extremely low latency, and massive data throughput, requirements that current 5G implementations struggle to consistently deliver [2–5].

A pivotal enabler of this transformation is network slicing, a technology already fundamental to 5G but envisioned to play an even more critical role in 6G. Network slicing allows the creation of virtualized, application-specific network segments on shared infrastructure, optimizing resources for diverse use cases. In the context of 6G, slicing ensures that resources are dynamically allocated, meeting the requirements in terms of latency, bandwidth, and security for each type of traffic. Beyond efficiency, slicing also bolsters security by isolating sensitive data flows within dedicated virtual networks, mitigating risks of data breaches or unauthorized access [1,5].

With the evolution of IoT driving unprecedented demand for adaptable and resilient networks, 6G and advanced network slicing promise to deliver a hyper-flexible, intelligent framework. This ensures not only seamless connectivity but also the scalability and security necessary to support the intricate web of devices and applications in the data-driven era. In this context, leveraging machine learning for network slicing selection emerges as a crucial strategy to optimize performance, address heterogeneity, and tackle the rising complexity of future communication systems [1,5,6].

As stated before, 5G could not deliver the stringent requirements for the future services that are expected to be delivered in the 6G era. Thus, Table 1 was created to better illustrate the differences between the minimum service requirements in 5G compared to those expected in 6G.

Table 1. Minimal requirements for 5G and 6G services [7,8].

5G Service	5G	6G
Peak Data Rate	Downlink: 20 Gbps Uplink: 10 Gbps	Downlink: ≥ 1 Tbps
Peak Spectral Efficiency	Downlink: 30 bps/Hz Uplink: 15 bps/Hz	Downlink: 300 bps/Hz
User Experienced Data Rate	Downlink: 100 Mbps Uplink: 50 Mbps	Downlink: 1 Gbps
User Plane Latency	1 ms	10–100 μ s
Reliability	1×10^{-5}	1×10^{-9}
Connection Density	1M devices/km ²	10M devices/km ²

1.1. Network Slicing in 6G

A network slice is an independent, logically isolated virtual network that operates on shared physical infrastructure. Unlike traditional Quality of Service (QoS), slicing delivers a comprehensive chain of compute, storage, and networking resources; containerized network functions (CNFs); and security tailored to the needs of individual services. By dynamically provisioning these slices, operators can efficiently handle varying demands while maximizing resource utilization. This flexibility becomes even more vital in 6G, as the range and complexity of applications surpass those of 5G. As we progress toward the 6G era, network slicing continues to evolve as a critical enabler of diverse and demanding applications. Originally introduced with 5G, the concept of network slicing provides end-to-end virtualized networks that connect user equipment (UE) to applications, tailoring resources to meet specific service requirements. However, in 6G networks, the scope and complexity of network slicing are significantly enhanced to accommodate the stringent requirements of

new services such as Further-Enhanced Mobile Broadband (feMBB), Extremely Reliable and Low-Latency Communications (ERLLCs), Ultra-Massive Machine-Type Communications (umMTCs), Massive Ultra-Reliable Low-Latency Communications (mURLLCs), and Mobile Broadband Reliable Low-Latency Communications (MBRLLCs) [9]:

1. feMBB: Analogous to 5G's eMBB, feMBB addresses applications requiring massive data rates, such as 3D video streaming, ultra-high-definition (UHD) content, virtual reality (VR), or augmented reality (AR). These applications demand slices optimized for very high data rates [10].
2. ERLLC: An evolution of URLLC, ERLLC supports applications requiring extreme reliability and ultra-low latency. Use cases include telemedicine, remote surgery, industrial Internet, and real-time remote sensing, where even the slightest delay could result in catastrophic consequences [10].
3. umMTC: Building on mMTC, umMTC expands the capacity to handle ten times more connected devices per square kilometer than 5G. This capability supports applications such as massive IoT ecosystems and smart cities, requiring efficient slices for massive device density [10].
4. mURLLC: This combines the high device density of umMTC with the low latency and reliability of URLLC. Applications like remote health monitoring depend on such slices to ensure reliable communication for millions of devices while maintaining minimal latency [2].
5. MBRLLC: A fusion of eMBB and URLLC capabilities, MBRLLC caters to services requiring high data rates and ultra-low latencies. Use cases include autonomous vehicles, delivery drones, and Vehicle-to-Everything (V2X) communications, where both speed and reliability are paramount [2].

To support these enhanced services, 6G network slicing leverages advancements in technologies like Network Function Virtualization (NFV) and Software-Defined Networking (SDN), which played a foundational role in 5G. NFV enables the dynamic deployment and management of virtual network functions, ensuring scalability and high availability for slices. Meanwhile, SDN provides centralized control, allowing for the dynamic re-configuration of slices based on real-time traffic conditions and service demands [11]. 6G takes network slicing to the next level by introducing cutting-edge technologies to meet its diverse and demanding use cases. AI enables the automation and coordination of slices through real-time predictions, ensuring efficient resource allocation. For slices needing top-tier security, quantum-resilient communication integrates quantum encryption and error correction to ensure data integrity. The use of the terahertz spectrum and advanced modulation enables the ultra-high bandwidth and low latency needed for services like feMBB and MBRLLC. With these innovations, 6G slicing adapts dynamically to real-time demands, delivering unmatched performance for everything from immersive AR/VR to critical IoT systems and autonomous vehicles [9]. Our proposed concept for network slicing in 6G is illustrated in Figure 1.

Network slicing, as defined in [12], enables the creation of logical networks, each tailored to support specific communication services (CSs) based on their unique requirements, such as latency, data rates, and coverage. These logical networks, or Network Slice Instances (NSIs), are composed of one or more end-to-end Network Slice Subnet Instances (NSSIs), which group network functions (NFs) and resources into manageable units. The advancements in network slicing, as outlined in [13,14], introduce significant enhancements across both core networks and radio access networks (RAN). In the core network, features such as the Network Slicing Admission Control Function (NSACF) manage user registrations and PDU sessions per slice, while the Network Slice Simultaneous Registration Group (NSSRG) facilitates provisioning multiple slices to users. The introduction of the Slice

Maximum Bit Rate (S-MBR) parameter ensures adherence to QoS constraints, enforced by the Policy Control Function (PCF). On the RAN side, the Network Slice As Group (NSAG) mechanism optimizes slice-aware reselection and RACH configurations, improving security and reducing overhead. Moreover, slice-specific RACH configurations and multi-carrier resource sharing enhance service continuity and resource prioritization during shortages. Subsequent developments in [14] address issues such as rejected S-NSSAI registrations, roaming capabilities, and deployments with limited coverage, focusing on finer-grained orchestration to improve performance, security, and resource utilization. On the other hand, AI-driven orchestration, as highlighted in [10], represents a transformative shift from traditional rule-based network slicing to intelligent, adaptive, and automated resource management based on historical or current data in the network. Besides advantages such as correct prediction of the network slice, sufficient allocation of resources for the slice type depending on network parameters, or adaptation of the slice in case of failure, the purpose of AI in orchestrating network slicing is also to pre-allocate resources in advance based on certain patterns and history, thus further reducing the delay that is a stringent characteristic in 6G. This transition represents a shift from the existing reactive approach to resource management and network slice deployment toward a more proactive strategy.

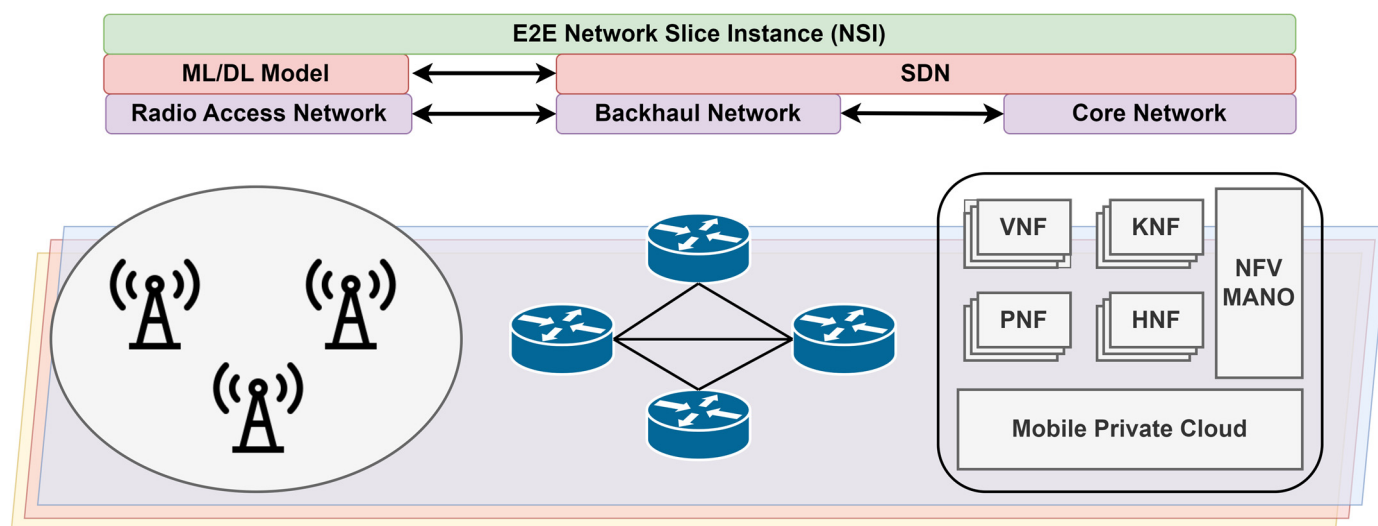


Figure 1. Proposed concept for network slicing in 6G.

1.2. Motivation and Contribution

As we will show in Section 2, most of the related works in the field of network slicing for 5G, Beyond5G (B5G), or 6G networks are based on relatively old datasets made strictly for LTE-A/5G such as the one provided by Crawdad [15]. Furthermore, other more general datasets include network traffic (not specific to mobile networks), such as Unicauca IP Flow Version2 [16]. In this paper, we propose a dataset for implementing the network slicing mechanism in 6G networks. Since the requirements for 6G are still in the standardization phase, we did not build the dataset in the Crawdad manner, by mapping use cases with QoS class identifiers (QCI or 5QI), as provided in 5G since Release 16 [17]. Our strategy was to collect the common traffic requirements proposed in several works [2,9,10,18] for future 6G services: feMBB, ERLLC, umMTC, mURLLC, and MBRLLC. We used them as features for training models based on Machine Learning (ML) and Deep Learning (DL) to recognize the appropriate slice for a given type of traffic and to perform the handover to a new slice in case the QoS parameters for the current slice degrade beyond a certain degree.

1.3. Outline of the Paper

This paper is organized as follows. Section 2 presents an overview of the state of the art for enabling network slicing in 5G/B5G/6G with the use of Machine Learning and Artificial Intelligence (AI). Section 3 describes the methodology and materials used to create the synthetic dataset and discusses the models involved in validating our approach. In Section 4 we present the experimental results obtained by each of our ML models and neural networks applied to the proposed dataset. The paper concludes by summarizing the contributions of the research and highlighting its significance.

2. Related Works

In the field of network slicing using ML and DL, numerous research studies have been conducted to achieve optimal slicing and meet the diverse requirements of various applications and services in 5G, B5G, and 6G networks. In this section, we present an overview of the existing literature, focusing first on studies that made contributions to the implementation of network slicing assisted by ML/DL.

The authors of [19] constructed a model with three main phases—data collection, Optimal Weighted Feature Extraction (OWFE), and slicing classification. They first built up a dataset consisting of attributes concerning several network devices, like user device type, duration, packet loss ratio, packet delay budget, bandwidth, delay rate, speed, jitter, and modulation type. The OWFE phase involved attribute value enhancement using a weight function that was optimized by the hybrid of the two meta-heuristic algorithms: the Glowworm Swarm Optimization (GSO) and Deer Hunting Optimization Algorithm (DHOA). The hybrid algorithm was called the Glowworm Swarm-based DHOA (GS-DHOA). A hybrid classifier was used in which Deep Belief Networks (DBNs) and neural networks (NNs) were fused with the weight function of the neural networks that was optimized by GS-DHOA for classification. The results of the experiments showed that the model was able to provide 5G network slicing precisely, outperforming Particle Swarm Optimization with Neural Networks (PSO-NN) + DBN, Grey Wolf Optimization with Neural Networks (GWO-NN) + DBN, GSO-NN + DBN, and DHOA-NN + DBN. The authors pointed out that their approach was effective but still had to be improved for more complicated problems in network slicing.

Another contribution to this topic comes from [20], which gave an investigation of the application of a multi-machine learning algorithm to improve 5G network slicing. It has thrown light on the fact that the 5G architecture itself supports very diversified service demands and multiple QoS requirements: latency, scalability, and throughput. It allows the network operators to create several virtual networks—the so-called slices—over the same physical infrastructure and manage them dynamically according to the operator and user-defined requirements. In this research, a variety of ML algorithms were used to train the model, to classify the network traffic, and to predict with high accuracy the type of slice for every user. It consisted of 66,000 rows and 9 columns, with KPIs as inputs and 5G slice types as output. In this paper, the performance of different ML algorithms has been compared based on the percentage of learnings, accuracy, precision, and F1 score. The accuracy of the model was 100%, making it quite effective in real-time traffic classification.

In addition, ref. [21] proposed two dynamic mechanisms that classify and send the traffic types through specific slices associated with 5G Quality-of-service Identifiers (5QI). In this work, the radio portion of cellular networks was the target. The radio resource sharing configurations that prioritize the critical data traffic were applied over a 5G standalone (SA) experimental network. One of the methods combined all flows via the most suitable slice, considering the dominant traffic type, while another one classified every data flow separately and transmitted them over the proper slice. The results have proven that network

slicing increases efficiency and reliability, especially for critical data, since it reduces packet loss and jitter. The second approach was more useful in heterogeneous traffic conditions, where greater control and accuracy over what is occurring is found, whereas the former found utility in resource-constrained or homogeneous traffic conditions.

In [22], a new data analytics tool is presented in support of cognitive resource management in 5G, called DeepCog. It tries to solve the challenge of forecasting resource demands for network slices by properly balancing the overprovisioning and service request violation trade-off. Unlike the traditional mobile traffic prediction models, it uses a deep learning architecture specially designed for capacity forecasting. DeepCog is the first tool that makes an analysis of the trade-offs between capacity over dimensioning and unserved demands in adaptive, sliced networks, thus showing the clear advantages of deep learning integration into resource orchestration. Comparative evaluations on real-world data proved the potential reduction in operating expenses by 50% or more against state-of-the-art traffic predictors, underpinning DeepCog's efficiency. The empirical results manifest the clear advantages that integrating deep learning into resource orchestration brings forth for a unique contribution of it to 5G systems on anticipatory resource management.

Moreover, the authors of [23] deal with the integration of network slicing and edge computing to meet the very heterogeneous and high QoS requirements of 5G and beyond. The paper discusses how network slicing prioritizes the virtualized logical networks over physical infrastructure, while edge computing allows real-time access to resources available at the network edge with high bandwidth and low latency. This paper provides a comprehensive survey on edge-enabled network slicing frameworks/solutions and illustrates some of its benefits through different use cases. It also touches on the application of machine learning in these edge-sliced networks and current advancements in the area, along with deployment scenarios. Another novel contribution of this work is that it proposes a reinforcement learning-based framework synchronizing controllers in distributed edge-sliced networks. It is toward this end that the framework seeks to improve the detection of optimal routing paths, balance resources, and make data offloading decisions to realize enhanced effectiveness of edge-enabled network slicing. The authors review how the fusion of machine learning with edge-enabled network slicing can realize scalable, flexible networks satisfying QoS/SLA requirements of emerging applications.

The research in [1] focuses on the performance enhancement for 5G networks using a new model called DeepSlice. This paper provides a solution for high reliability, low latency, high capacity, and security problems of 5G networks using network slicing and deep learning techniques. DeepSlice is a framework developed using a deep learning view on neural networks for improving efficiency in network load and availability, so far as analyzing the Key Performance Indicators (KPIs) that can predict and allocate network slices to device types that are unknown. This model allows intelligent system resources to be allocated and load-balanced to ensure efficient resource use across existing network slices while optimizing slice selection even in the case of network failures. The authors presented a study on the benefits of DeepSlice for correct prediction of the most appropriate network slice, given the device parameters, to aid in enhancement of the management of network loads and how to handle slice failures.

Another important contribution is given in [24], which suggests a hybrid model, capable of satisfying the dynamic requirements of 6G through the integration of a Convolutional Neural Network (CNN) with a Bidirectional Long Short-Term Memory (BiLSTM) network. This aims to improve the flexibility and accuracy of network slicing in the context of 6G. In this paper, the applicability of the hybrid model was tested using the Unicauca IP Flow Version2 dataset. In the model, a CNN was used for automatic feature extraction, while a BiLSTM was responsible for the classification of which network slices should be assigned.

Its overall recognition rate reached 97.21%, thus proving that the model is very good at offering a reliable and accurate network slice to the end user. In this study, the model performance is evaluated using stratified 10-fold cross-validation for the appropriate allocation of network slices to new traffic requests. This is the approach that solves the main challenge of mapping slices to unidentified devices, hence improving efficiency in network resource management in the 6G network.

A comparative summary of the works discussed in this section, including the results from the present study, is presented in Table 2.

Table 2. Overview of network slicing research: AI/ML algorithms and performance metrics.

Paper	Technology	AI/ML	Results	Intended Problem
[1]	5G	RF and DL (NN)	95% accuracy	Network slicing classification and load balancing prediction
[19]	5G	Hybrid GS-DHOA + DBN, NN	Outperform PSO-NN + DBN, GWO-NN + DBN, GSO-NN + DBN, and DHOA-NN + DBN	Network slicing optimization
[20]	5G	Naïve Bayes, SVM, NN, Gradient Boosted Tree, RF	100% accuracy	Real-time traffic classification for network slicing
[22]	5G	Deep Neural Network (DNN)	50% reduction in operational expenses	Resource management in network slicing
[24]	6G	CNN + BiLSTM	97.21% accuracy	Network slicing classification
Current	6G	Naïve Bayes, SVM, kNN, RF, Decision Trees, XGBoost, Feedforward Neural Network (FNN)	99% accuracy	Network slicing classification and load balancing prediction

3. Materials and Methods

Machine learning and neural networks are widely used in all industrial applications and were also incorporated in 5G networks. Their usage will grow even more rapidly in 6G due to the massive data generated by the devices, faster processing time requirements, and faster decision making by the management systems. As shown in Table 3 and already presented in Section 1, in 6G, the services will be more granular than in 5G, where the main services were eMBB, URLLC, and mMTC. In this study, we identified five proposed categories of services: feMBB, umMTC, mURLLC, ERLLC, and MBRLLC. Accordingly, our objective is to dynamically allocate traffic in a more granular manner and establish distinct slices tailored to each of these use cases.

In the current work, we analyze multiple QoS parameters, and we used them to train different models to predict the network slice for a specific type of traffic. Since we could not find a suitable dataset for our needs, we generated the data based on [2,9,10,18], which consists of 10,000 unique input combinations. The parameters used to train the model are those presented in Table 3: Packet Loss (Reliability), Latency Budget, Jitter Budget, Data Rate Budget, Required Mobility, and Required Connectivity. We assume that these parameters can be monitored and captured from the packets exchanged by the UE with the 6G network and then injected into our model in real time. Of course, once standardized, the 6G QoS Identifiers (6QI) could integrate our features as in 5G, and the dataset could be updated. Our main goal in this paper is to demonstrate that using the found proposed

parameters related to the traffic requirements, the slice can be assigned appropriately based on AI. The dataset can be found in [25].

Table 3. Features of the training models for slice selection.

Use Case	Packet Loss	Latency	Jitter	Data Rate	Mobility	Connectivity	Predicted Slice
3D/UHD Video	$<10^{-5}$	ND	ND	>1 Tbs	No	No	feMBB
VR/AR	$<10^{-5}$	ND	ND	>1 Tbs	No	No	feMBB
Smart City	ND	ND	ND	ND	No	Yes	umMTC
Remote Health Monitoring	$<10^{-5}$	<1 ms	ND	ND	No	Yes	mURLLC
Servo Motors	$<10^{-7}$	<0.1 μ s	<1 μ s	ND	No	Yes	ERLLC
Motion Control	$<10^{-6}$	<1 μ s	<1 ms	ND	No	Yes	ERLLC
Delivery Drones	$<10^{-6}$	<1 ms	ND	ND	Yes	Yes	MBRLLC
V2X	$<10^{-9}$	<0.1 ms	ND	ND	Yes	Yes	MBRLLC

For services for which we did not find strict requirements, we initially used the 5G limit values. For example, we did not find clear latency budget requirements for feMBB or umMTC services, so we considered the maximum allowable latency to be that corresponding to URLLC services in 5G, 4 ms. We made a similar analogy regarding the jitter. For services that did not have strict requirements regarding it, we set a maximum allowed value of 4 ms. Although there are requirements for the transfer rate for feMBB services, namely 1 Tbps, MBRLLC services must still meet the requirements of eMBB and URLLC in 5G. Therefore, we considered that the budget for services that do not require the transfer rate should be any value less than 1 Tbps, that for MBRLLC should be between 100 Gbps and 1 Tbps, and at least 1 Tbps for feMBB. Finally, we have mobility and connectivity requirements, which can be conveyed by a specific bit. Even if, according to [18], V2X applications would have higher mobility requirements than delivery drones, both cases fall under the MBRLLC slice. Analogous to the applications that require high connectivity, these are similar, the difference between umMTC and mURLLC services being the delay requirements. Subsequently, we considered that the parameters that do not yet have (have not been found) defined requirements should be mapped with the same value, and for simplicity, this should be 0. This mapping does not affect the implemented models, since it is the same for the services that did not have the respective traffic requirements defined.

Another use case of our work is related to the handover operation that the operator must perform if the quality in terms of QoS parameters for a current slice degrades. For this, we considered 4 input features, namely Slice Available Transfer Rate (SATR), Slice Latency (SL), Slice Packet Loss (SPL), and Slice Jitter (SJ). Based on these parameters, we trained our models to predict the Slice Handover (SH) operation (see Table 4 and Figure 2).

Table 4. Features of the training models for slice handover.

SATR	SL	SPL	SJ	SH
1 Gbps–10 Tbps	1 ns–4 ms	0 – 10^{-3}	1 ns–4 ms	Yes/No

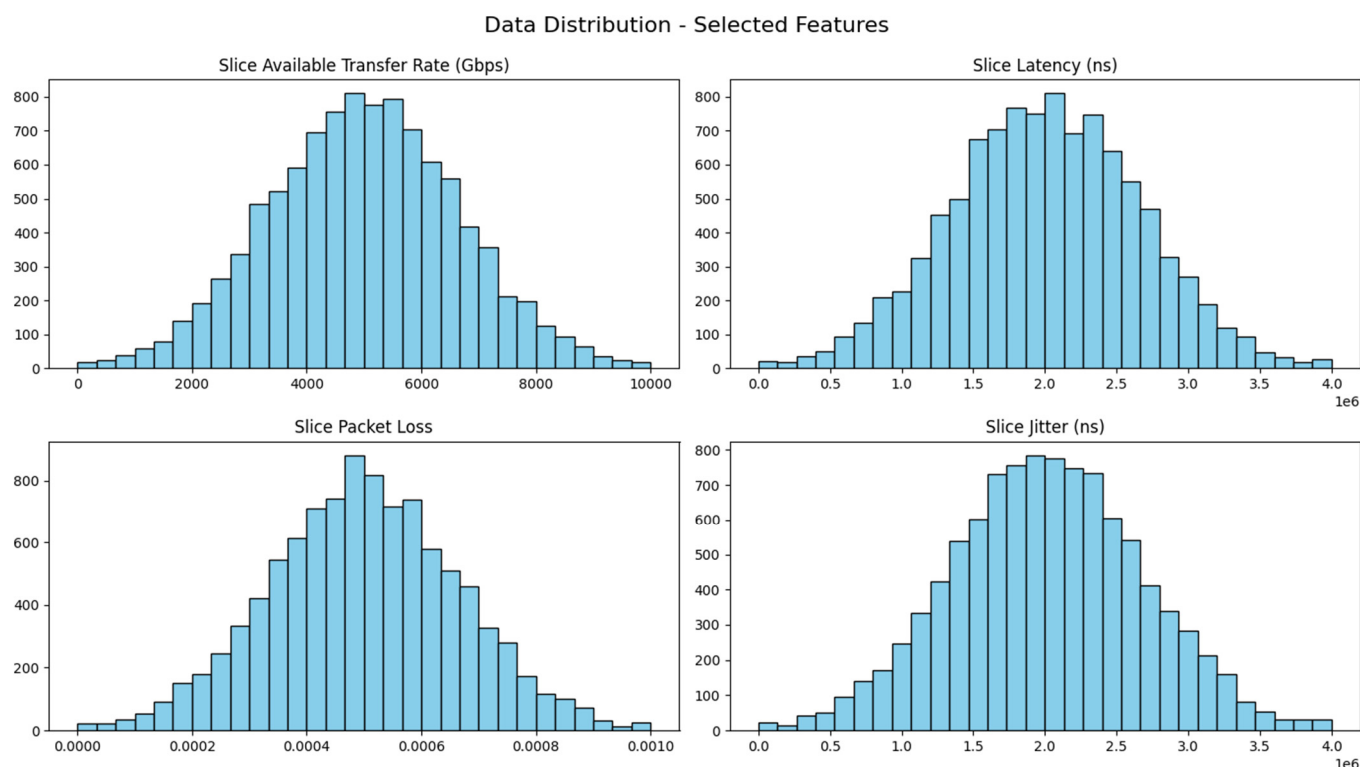


Figure 2. Data distribution for slicing QoS parameters.

These network parameters can be measured in real time and fed to the model to estimate whether traffic on the current slice should be moved to a new one. This AI-based approach is also a natural continuation of our previous work [7,11] in which we performed network slice selection using deterministic and heuristic methods. Handover to a new slice must be performed if the budget regarding these parameters required by a service does not correspond to the current values in the slice. The dataset's parameters, including slice latency, packet loss, jitter, and available transfer rates, were generated using a Gaussian distribution constrained within realistic ranges to simulate diverse 6G network conditions, as shown in Figure 2.

All the experiments in this paper were conducted on a server with an Intel Core i7 10th generation, 32 GB of RAM, and an Nvidia RTX 3070 dedicated GPU.

3.1. Machine Learning Models

The data was trained by splitting the original dataset into 80% training data and the rest for testing. Several machine learning techniques were used in this paper to implement the network slicing selection.

a. Support Vector Machine

The Support Vector Machine is a supervised learning algorithm that is applicable to both classification and regression tasks. It is particularly effective for binary classification, where it distinguishes between two classes. The model takes as input a set of pre-labeled messages and generates a hyperplane that separates the two classes. SVM is regarded as one of the most reliable classification models for datasets with a limited amount of labeled data.

b. Decision Trees

Decision Trees offer several advantages, notably their resilience to outlier values, as well as their capacity to manage missing data and work effectively with categorical

variables. The decision tree algorithm determines the optimal data splits at each node by evaluating criteria such as information gain or Gini impurity, both of which quantify the quality of a split. Information gain measures the reduction in entropy, indicating how well a split separates classes, while Gini impurity assesses the probability of misclassification within a subset. By systematically applying these criteria, the algorithm identifies the most informative attributes, using them to partition the data iteratively and structure the tree's branching architecture.

c. Random Forest

Random Forest (RF) is a widely used supervised ML algorithm suitable for both classification and regression tasks. The Random Forest classifier is a decision tree-based algorithm that operates as an ensemble classifier, comprising multiple decision trees of varying sizes. Ensemble learning methods, such as RF, improve predictive performance by combining multiple algorithms to produce more accurate results than individual models alone. It is widely recognized as one of the most effective supervised classification techniques based on ensemble learning, thanks to its robustness against overfitting and its unique tree-combining rules. RF employs the Bagging (Bootstrap Aggregation) technique, an ensemble method that enhances performance by combining multiple models. Bagging involves selecting random samples from the dataset; each model is then trained on these samples, created through replacement (row sampling), a process known as bootstrapping. It constructs multiple decision trees on different subsets of data samples, using majority voting for classification tasks and averaging for regression. Each tree is trained independently, and their individual outputs are aggregated to generate the result. For classification, this involves majority voting across all models. In our implementation, we used the `RandomForestClassifier` class from the `sklearn` library to build the Random Forest model.

d. Gaussian Naïve Bayes

Gaussian Naïve Bayes (GNB) is a supervised machine learning algorithm grounded in Bayes' Theorem, which calculates posterior probabilities based on conditional probabilities under the assumption that features are independent of one another. The Gaussian variant of Naïve Bayes employs a specific probability distribution, represented by Equation (1):

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \quad (1)$$

In this equation, σ_y denotes the standard deviation for class y , while μ_y is the mean value for class y ; both parameters are estimated through maximum likelihood. For our implementation, we utilized the `GaussianNB` class from the `sklearn.naive_bayes` module in the Scikit-learn library.

e. kNN

The k-Nearest Neighbors (kNN) algorithm is a simple, yet effective, supervised machine learning method used for both classification and regression tasks. This non-parametric algorithm is commonly applied in classification tasks, as it does not rely on any assumptions regarding data distribution. The k-Nearest Neighbors (kNN) algorithm is a data classification technique used to estimate the probability that a given data point belongs to a particular group based on the group memberships of its nearest neighbors.

kNN is an instance-based learning algorithm that operates by storing all available cases and classifying new instances based on similarity measures, typically Euclidean distance. Given a new data point, the algorithm identifies the k closest training instances,

or neighbors, and assigns the label that is most common among them (for classification) or calculates the average of the labels (for regression).

The choice of k , the number of neighbors considered, significantly impacts the model's performance: a small k may lead to overfitting, capturing noise in the training data, while a larger k tends to smooth the decision boundary but may oversimplify complex patterns. Additionally, the kNN algorithm is non-parametric, meaning it does not assume an underlying probability distribution for the data, making it well-suited for applications where the data may not conform to known statistical distributions. Despite its simplicity, kNN is computationally intensive, especially with large datasets, since it calculates the distance between the new data point and every instance in the training set. We used the `KNeighborsClassifier` class from the `sklearn` library to construct the kNN model in our implementation.

f. XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and efficient gradient boosting algorithm commonly used for classification and regression tasks. It operates by sequentially training decision trees, where each new tree corrects the errors of the previous ones, improving the model's performance. XGBoost incorporates both L1 and L2 regularization to prevent overfitting, making it more robust and generalizable compared to traditional gradient boosting methods. Additionally, it supports automatic handling of missing data and parallel processing, allowing for faster model training, especially with large datasets.

3.2. Deep Learning Model

Multi-output Feedforward Neural Networks (FNNs) offer an extension of traditional neural networks by allowing simultaneous prediction of multiple targets. These models are particularly effective when the tasks share underlying patterns, as they can learn a common set of features through shared layers and then refine these features for each specific task in separate output layers. In this study, we utilized a multi-output FNN implemented with the TensorFlow framework and Keras to tackle a network slicing problem, predicting two related outputs: Slice Type and Slice Handover.

The model begins with an input layer that processes the normalized features of the dataset. This is followed by two shared dense layers that act as the backbone of the network, extracting general patterns from the input data. The first dense layer contains 64 neurons with a Rectified Linear Unit (ReLU) activation function, which is well-suited for capturing non-linear relationships. A second dense layer with 32 neurons and ReLU activation further refines these features. The shared representation is then split into two task-specific output layers. Each layer corresponds to one of the two target variables: (a). Slice Type: a dense layer with a softmax activation function, outputting a probability distribution over the possible categories of network slices, and (b). Slice Handover: a similar dense layer with a softmax activation, providing the likelihood of each category in the handover requirement. This architecture ensures that the shared layers learn common features useful for both tasks, while the output layers are tailored to their specific prediction requirements.

The model was trained using the Adam optimizer, which is widely regarded for its ability to adapt learning rates dynamically, making it effective for complex optimization problems. Categorical cross-entropy was chosen as the loss function for both targets, as it is particularly suitable for multi-class classification tasks. To assess performance, accuracy was used as the primary metric for both outputs. Data preparation included splitting the dataset into training (80%) and validation (20%) sets. Input features were standardized using a `StandardScaler`, which scales the data to have a mean of zero and a standard deviation of one, ensuring consistent model convergence. The model was trained

over 30 epochs with a batch size of 32. Validation data was used to monitor performance throughout the training process.

3.3. Models Evaluation

To evaluate each ML model's performance, we calculated the following:

1. Accuracy: The proportion of correct predictions. The accuracy calculation formula can be found in Equation (1) and is identified as the ratio between the sum of the true positive predictions (TP) and the true negative ones (TN) and the sum of the two previously specified, to which the false positive predictions (FP) are added and false negatives (FN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

2. Precision: The proportion of positive predictions that were correct.

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

3. Recall: The proportion of actual positive cases that were correctly identified.

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

4. F1-score: The harmonic mean of precision and recall, providing a balanced measure of performance.

$$F1_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (5)$$

5. **Support:** The actual occurrence of a class in the dataset.

Then we generated the confusion matrix for the test data. Moreover, we generated a second dataset to estimate how our model behaves on new, unseen data. For the new dataset, the above metrics were recomputed, and the confusion matrix was regenerated. The confusion matrix encapsulates the 4 important elements that determine the accuracy of the model, namely TP, TF, FP, and FN, and summarizes them by comparing the real labels with those predicted by the algorithm. As a composition, the main diagonal of the matrix contains only values that were classified as positive (TP or TN); everything below the main diagonal is elements classified as false negatives, and what lies above the main diagonal are false positive classifications.

As for the FNN model, we generated accuracy and loss learning curves over the epochs and the confusion matrices. Loss quantifies the difference between the model's predictions and the actual target values. In this case, categorical cross-entropy was used as the loss function for both Slice Type and Slice Handover, which is calculated as

$$Loss = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log \hat{y}_{ij} \quad (6)$$

where N is the number of samples, C is the number of classes, y_{ij} is the true label, and $\hat{y}_{ij}$ is the predicted probability for class j of sample i . A decreasing trend in loss indicates that the model is improving its predictions over time, while significant differences between training and validation loss might suggest overfitting.

4. Experimental Results

To assess the model's predictive performance and generalization, a stratified 5-fold cross-validation was employed. This technique ensured that the distribution of classes was preserved across different folds, mitigating potential biases due to class imbalance. By dividing the dataset into five equal parts, each model was trained in four folds and tested on the remaining fold. This process was repeated five times, with each fold serving as the test set exactly once. For each target variable, we assessed the models' accuracy, which measures the proportion of correctly classified samples. The mean accuracy scores across the five folds were then computed, providing a single performance measure for each model on each target. The results of this analysis, including the mean cross-validation scores for each model and each target variable, are presented in Table 5. Note that we removed Use Case, Jitter, Mobility, and Connectivity from the list of features in Table 3 because they were too correlated with the output and had too much importance in prediction.

Table 5. Mean cross-validation scores obtained for implemented ML models.

Mean Cross-Validation Score	Random Forrest	Decision Trees	Naïve Bayes	SVM	kNN	XGBoost
Slice Type	1.0	0.88	1.0	0.88	0.84	1.0
Slice Handover	0.9995	1.0	0.77	0.88	0.84	0.9995

It is important to note that these preliminary results were obtained without performing hyperparameter optimization. We will first present these preliminary results and then discuss the results obtained after applying the previously mentioned strategy.

Next, we calculated the accuracy, precision, recall, and F1-score for the validation set, and generated plots to illustrate them for both targets: Slice Type (see Figure 3) and Slice Handover (see Figure 4).

It can be observed that the values obtained for accuracy are correlated with the ones obtained for cross-validation scores. The model obtained using the random forest classifier, even with the default parameters, managed to obtain an almost perfect score in predicting both targets, while the other models have satisfactory results, depending on what output they need to predict. For example, the decision tree-based model presents an almost perfect score for all parameters (over 99%) when it must predict the handover moment but has difficulties in predicting the slice type (more precisely, the mURLLC class is confused with MBRLLC, according to Figure 5, where the confusion matrix obtained for the validation set for this model is illustrated).

What can be observed, based on Figure 5 and Table 3, is that even if the entire mURLLC class is confused with MBRLLC, this error will not induce a degradation of the services classified on that slice, because MBRLLC has more rigorous traffic requirements than mURLLC. Obviously, it could represent a problem if the operator does not currently have the resources necessary to allocate this type of slice. Note that in this case, a max_depth parameter had the value of 3 (discussed in Section 4.1).

The model using the Naïve Bayes classifier behaves somewhat in the opposite way to the Decision Tree model. Thus, it manages to classify the slice type almost perfectly but has satisfactory accuracy regarding when services should be moved to another slice, having a score of 82% for accuracy, 80% for precision, 88% for recall, and 80% for the f1 score. Figure 6 represents the confusion matrix generated for the validation set of the model that used the Naïve Bayes classifier. The difference from the previous model is that it can induce service degradation when the prediction is wrong, because approximately 23% of the services that will need a new slice due to congestion will be left on the same slice.

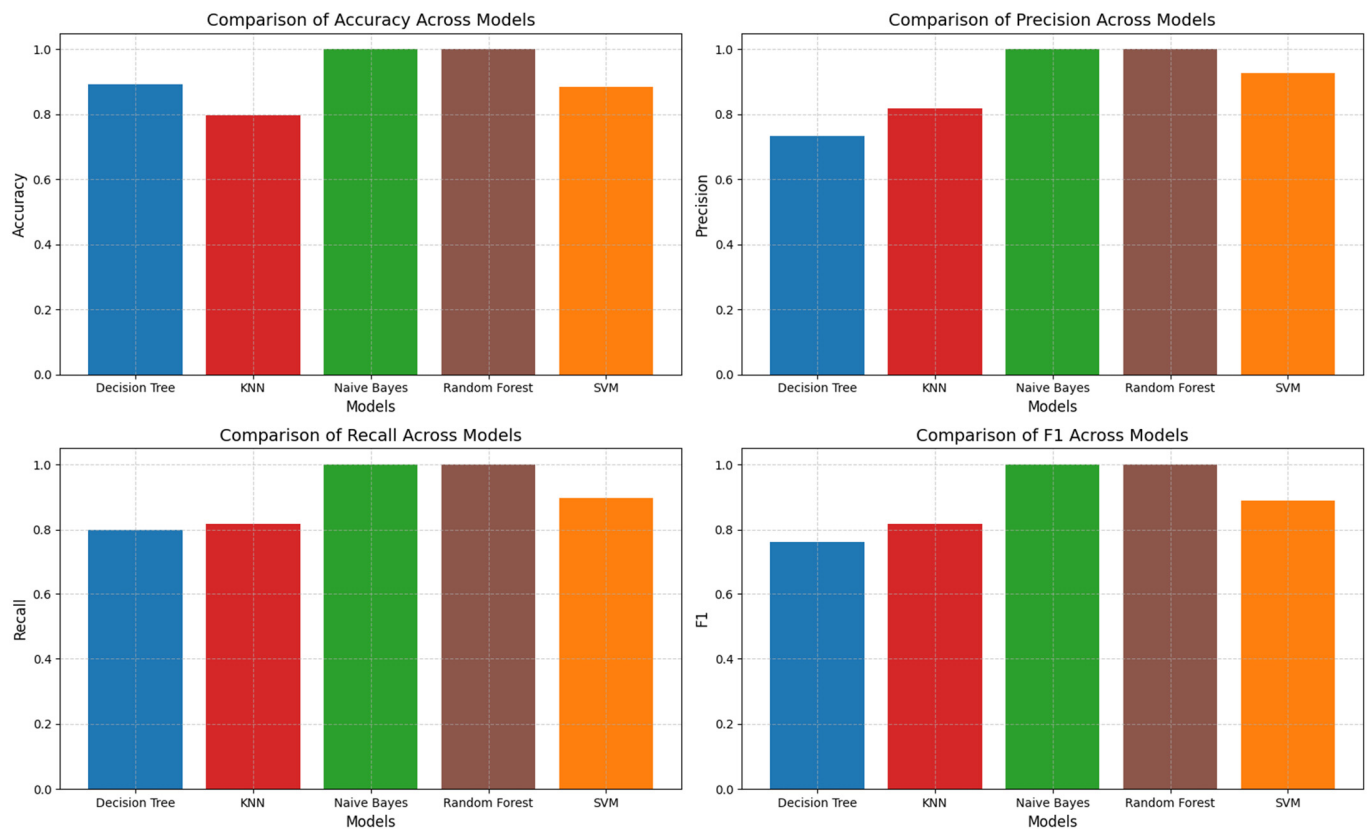


Figure 3. Model evaluation for slice type target.

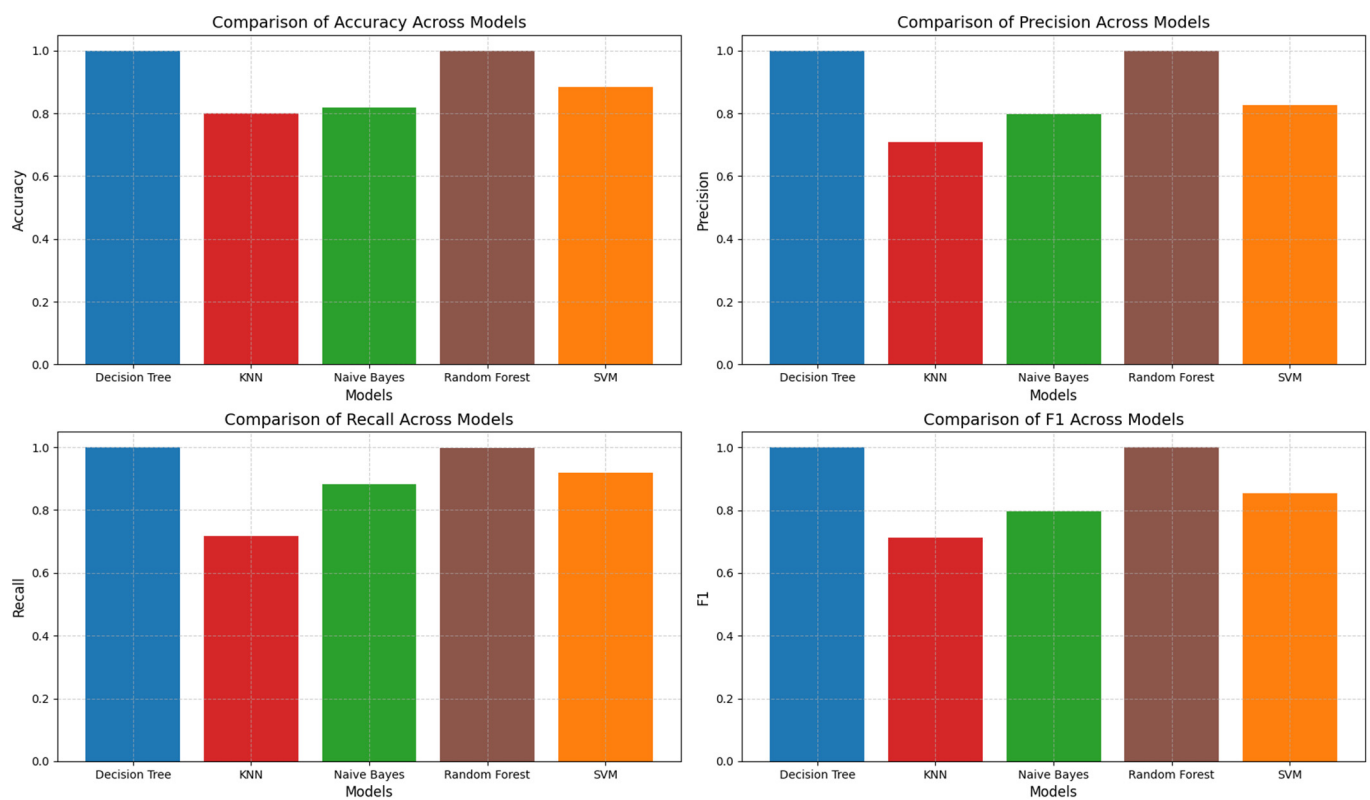


Figure 4. Model evaluation for slice handover target.

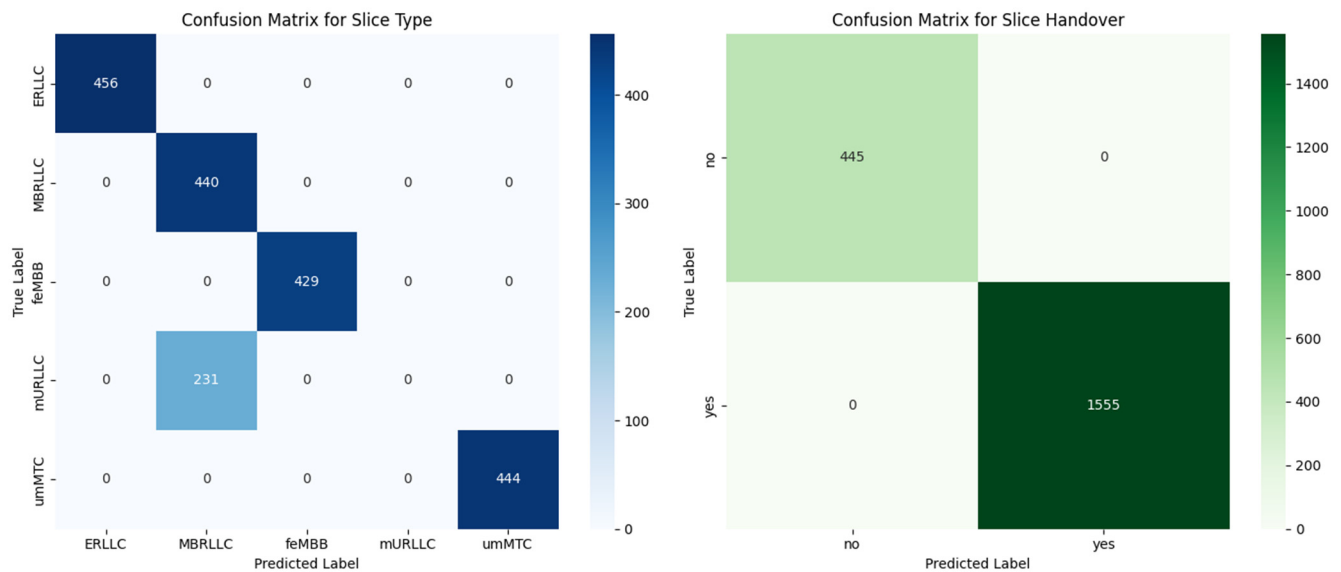


Figure 5. Confusion matrix for the validation set of the decision tree model.

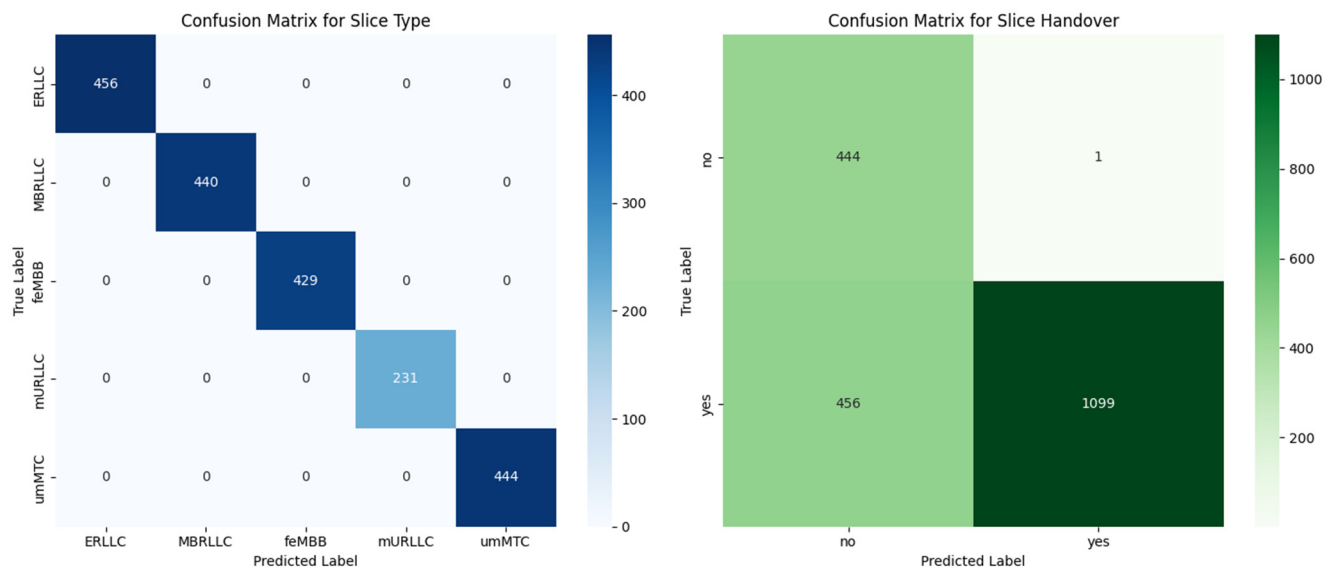


Figure 6. Confusion matrix for the validation set of the Naïve Bayes model.

The remaining models, namely those in which the SVM and kNN classifiers were used, are presented herein together, as they have common features. The confusion matrices are presented in Figure 7 (for SVM) and Figure 8 (for kNN). The scores for accuracy, precision, recall, and F1 for slice handover are 0.8, 0.71, 0.72, and 0.71 (for kNN), respectively, and 0.88, 0.83, 0.92, and 0.85 (for SVM). For slice type, the scores for kNN are 0.8, 0.81, 0.81, 0.82, and, respectively, 0.88, 0.93, 0.9, and 0.89 for SVM. The results obtained for accuracy on the validation set are close and slightly lower than those obtained for the cross-validation score.

It can be seen from Figures 7 and 8 that the two models have the same issue, namely in clearly distinguishing between ERLLC and umMTC slices. Moreover, the handover moment has a much worse accuracy than in the case of previous models. Unfortunately, these errors are of great severity because ERLLC services have very strict traffic requirements compared to umMTC. Moreover, these models do not perform better in terms of the decision to execute handover either. The issue with the decision to perform a handover, as we specified above, is more critical when postponing the operation when needed than performing an additional handover. From the confusion matrices, we can see that these

false negative cases are in a relatively high proportion, 10.7% for SVM and 7.75% for kNN, suggesting that it would not be reliable to implement these models on a real testbed.

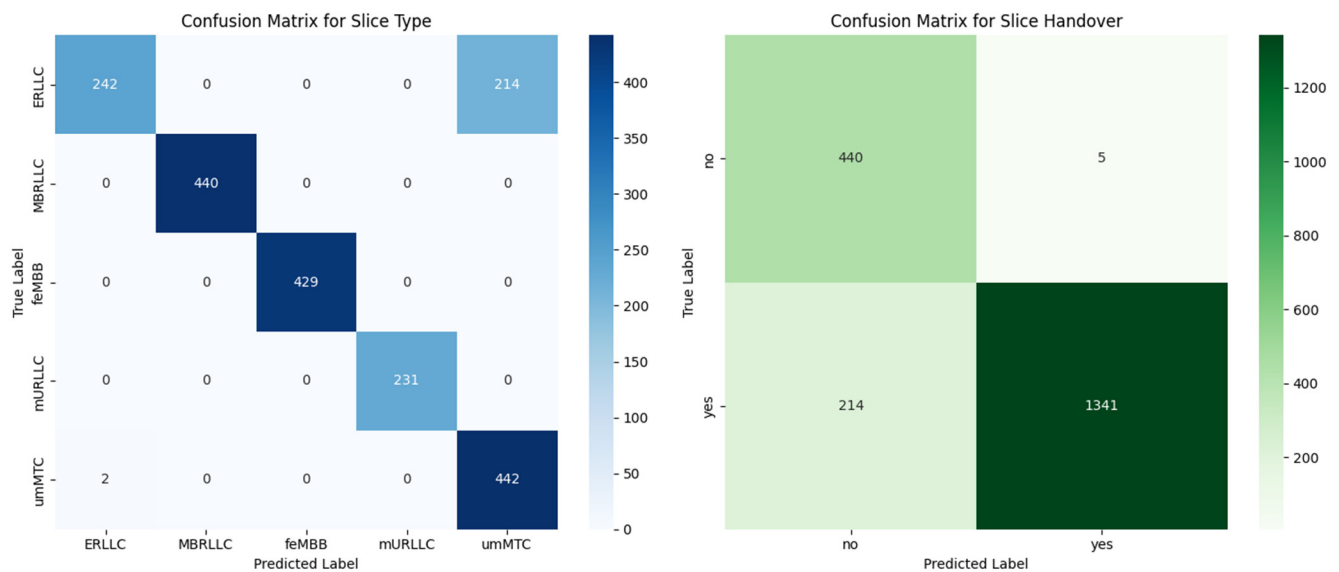


Figure 7. Confusion matrix for the validation set of the SVM model.

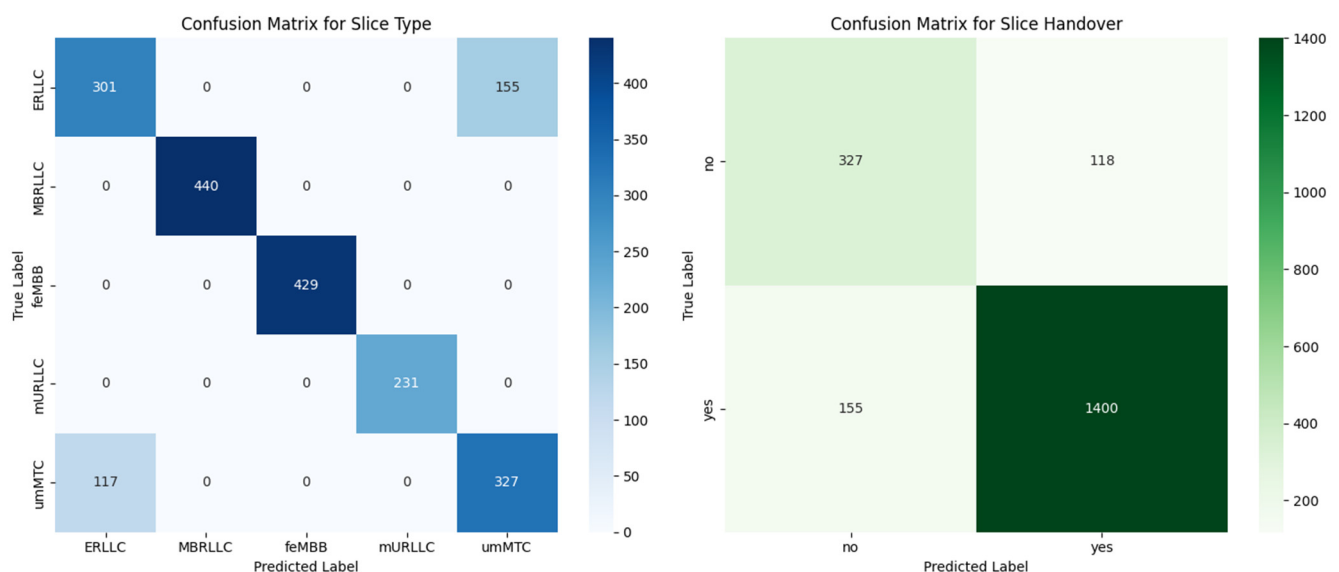


Figure 8. Confusion matrix for the validation set of the kNN model.

4.1. Hyperparameter Tuning

In this paragraph, we performed various operations, such as GridSearch, using the GridSearchCV function from the sklearn library for hyperparameter tuning to try to increase the accuracy of kNN and SVM models. For the model based on a Decision Tree, we analyze the impact with the variation of the parameter `max_depth`.

a. Decision Tree

The results in Table 5 highlight the very good performance of Random Forest and Decision Trees, both of which achieved near-perfect scores across the two tasks. This underlines their ability to effectively uncover complex, nonlinear relationships in the data. Specifically, the Random Forest algorithm showed significant robustness; its performance was very high for all the different values of `max_depth` used, provided that the latter was greater than 1. Decision Trees also produced good results, with optimal performance at

max_depth equal to 5, where the model balanced the complexity of the data while reducing overfitting. However, reducing the depth below 5 led to a performance decrease, likely due to underfitting, as the shallower trees could not capture the subtleties in the data. For example, for a value of max_depth of 3, the mean cross-validation score dropped to 0.8921 for the Decision Tree model. The max_depth parameter is important in any tree-based model as it limits the longest path from the root to any leaf. A higher value of max_depth can capture more complex details. However, this can result in overfitting on the training dataset. Conversely, smaller values may help prevent overfitting, but if too small, it may result in an overly simple model and underfitting.

b. kNN

For the KNeighborsClassifier we conducted the search by varying three parameters: metric, n_neighbors, and weights. The weights could be uniform or distance, metric Euclidean, Manhattan, or Minkowski, and we increase the number of neighbors from 1 to 9 with a step of 2. The n_neighbors parameter defines how many nearest neighbors are considered when making predictions; smaller values make the model sensitive to noise and prone to overfitting, while larger values can smooth out predictions but may underfit. The metric parameter specifies the distance function used to measure the proximity between data points and the weights parameter controls how neighbors contribute to predictions, with uniform weights treating all neighbors equally and distance weights giving closer neighbors more influence, allowing the model to focus more on local patterns in the data. In Figure 9 we present the variation in accuracy with increasing number of neighbors for the three distances in two subgraphs: the one on the left corresponds to the weight parameter being uniform, and the one on the right, distance.

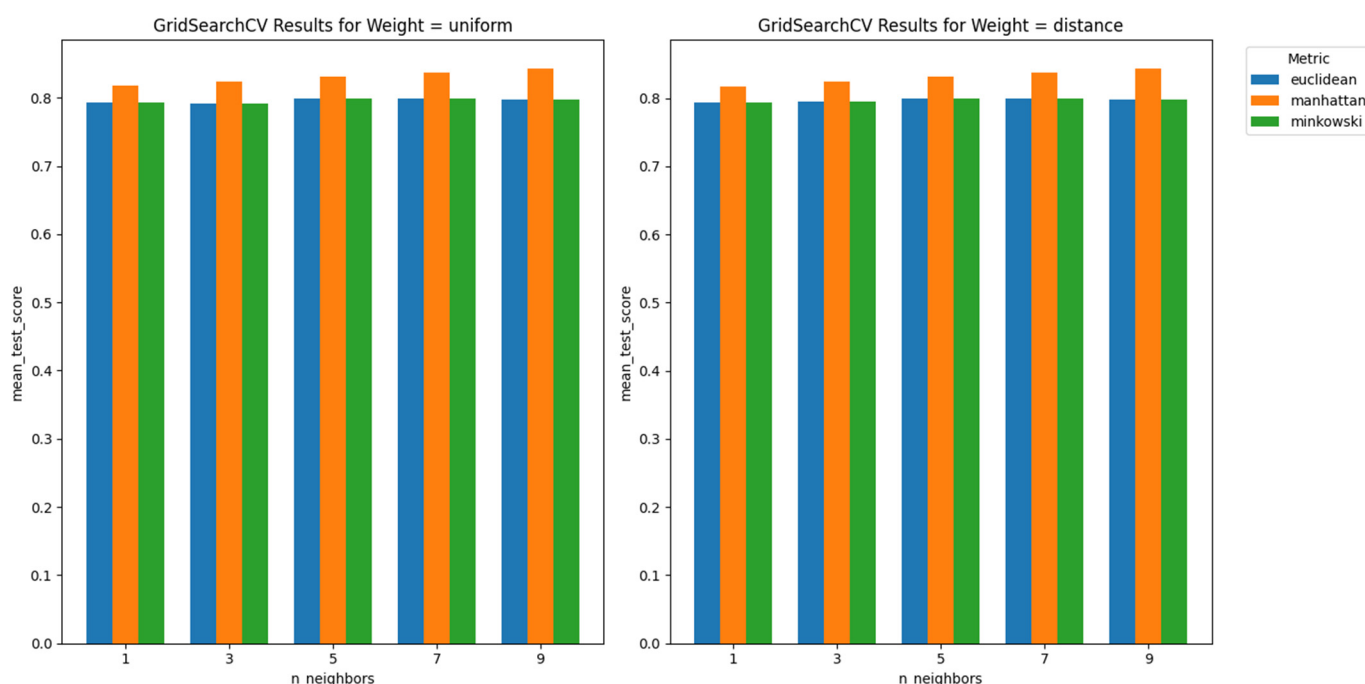


Figure 9. kNN model hyperparameter tuning.

The lowest score of 79.23% corresponds to the configuration formed by the Euclidean metric, with three neighbors and uniform weight, and the highest, of 84.33%, corresponds to the configuration formed by the Manhattan metric with nine neighbors and distance weights. Adding more neighbors does not necessarily help increase accuracy, as can be seen in Figure 10. Adding more neighbors does not necessarily help increase accuracy, as can be seen in Figure 10, where we increased the number of neighbors from 1 to 21 in steps

of 2. A maximum is observed for the number of neighbors equal to 15, after which the accuracy starts to decrease. Moreover, the difference in accuracy from several neighbors of 7 to 15 is less than 0.5%; therefore, increasing the number of neighbors above 7 does not bring a major benefit to the model. Furthermore, we observed that although Euclidean (81%) and Minkowski (80.3%) distances offer roughly the same performance, using the Manhattan distance improves the model accuracy by almost 5 percent (86.41%). The results are illustrated in Figure 11 and are performed for several neighbors equal to 9 and distance weights. The observed performance, where Manhattan distance outperforms Euclidean distance and Minkowski distance, can be attributed to how each distance metric interacts with the scale and discrepancies in feature values. Manhattan distance, calculated as the sum of absolute differences between features, is less sensitive to large discrepancies in feature values, making it more robust when features have differing magnitudes. This is particularly important when features, even after scaling, retain large value discrepancies, as is the case in the dataset. Euclidean distance, on the other hand, calculates the “straight-line” distance between points, which amplifies the effect of large differences through squaring. This makes Euclidean distance more sensitive to discrepancies in feature values, especially when features differ by large orders of magnitude. As a result, when features are not perfectly scaled or have wide value ranges, Euclidean distance can lead to suboptimal performance. Minkowski distance generalizes both Manhattan and Euclidean distance by introducing a parameter p . When $p = 1$, it behaves like Manhattan distance, but as p increases, it becomes more sensitive to larger differences in feature values, similar to Euclidean distance. This results in the observed decrease in accuracy as p increases, as the metric becomes more influenced by features with extreme values.

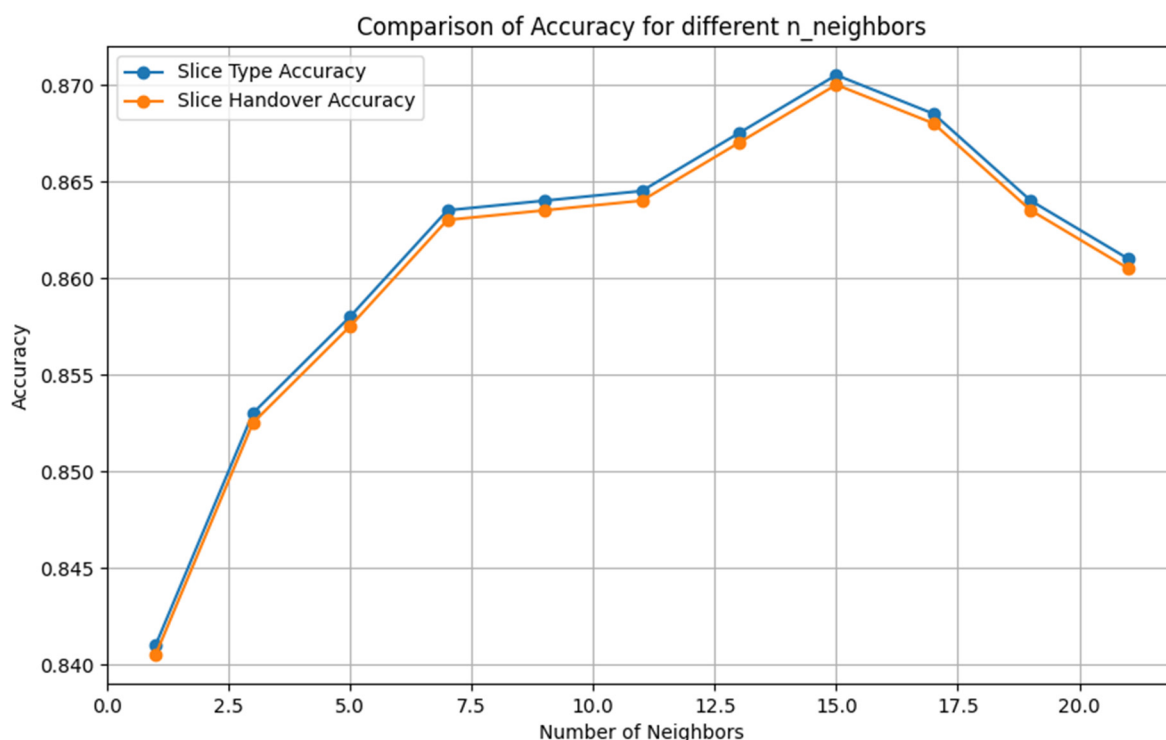


Figure 10. Accuracy variation with increasing number of neighbors.

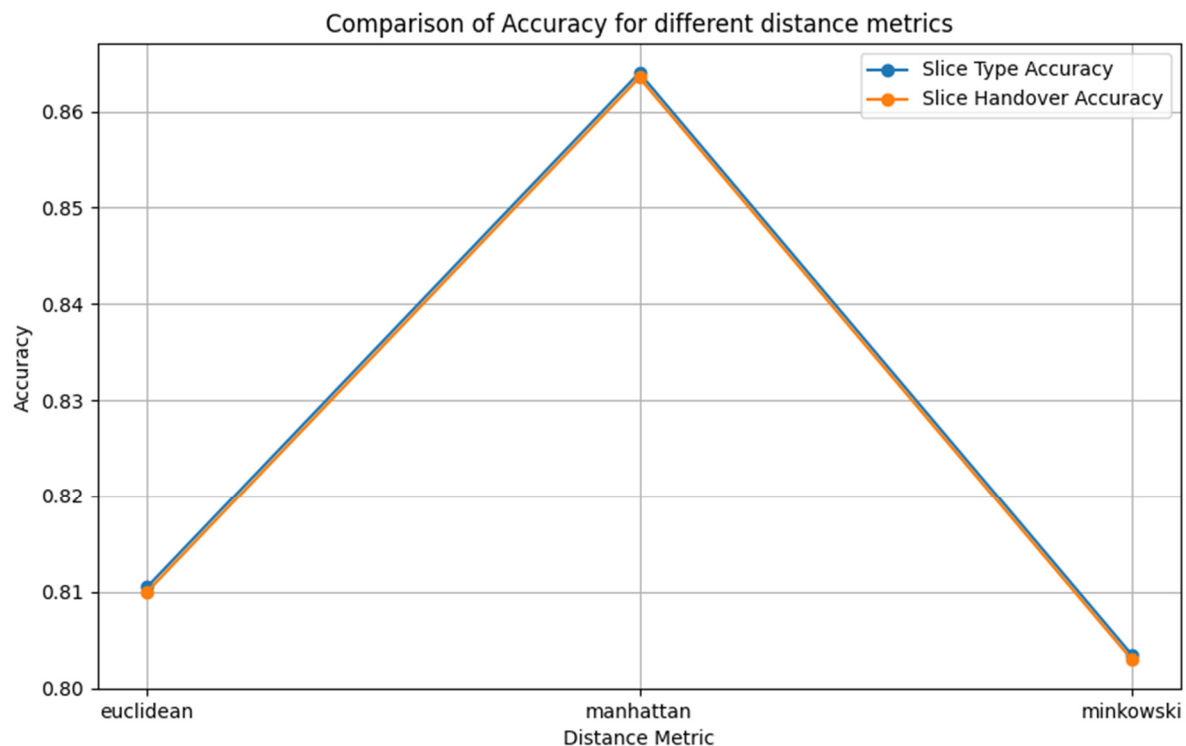


Figure 11. Accuracy comparison of different distance metrics.

c. SVM

In Support Vector Machine, the kernel, C , and gamma parameters are crucial for defining the model's performance and its ability to generalize. The kernel function is used to transform the input data into a higher-dimensional space, allowing SVM to find complex, non-linear decision boundaries. Common kernels include the Radial Basis Function (RBF), polynomial, and linear kernels, each serving to map the data into spaces where it is more likely to be linearly separable. The C parameter controls the trade-off between achieving a low training error and maintaining a simple model. A large value of C emphasizes minimizing misclassification on the training set, potentially leading to overfitting by producing a more complex decision boundary. Conversely, a smaller C allows for more misclassifications, promoting a simpler, more generalized model. The gamma parameter, which is particularly important when using the RBF kernel, defines the extent of influence a single training point has on the decision boundary. A high value of gamma results in a more localized influence, leading to a highly complex decision boundary that risks overfitting, while a lower value promotes a smoother boundary, which may underfit the data.

We performed the cross-validation using four different kernels: linear, rbf, poly, and sigmoid. For the first three, we got the same average cross-validation score of 88% for both outputs and using the rest of the parameters at default values. For the sigmoid kernel, however, the score dropped drastically to almost 80%. Next, we tested the best configuration found by GridSearch, namely for the rbf kernel, with the value of C set to 100 and gamma to 0.1. For this configuration, we managed to increase the model accuracy to over 99%, as can be seen from the confusion matrix in Figure 12.

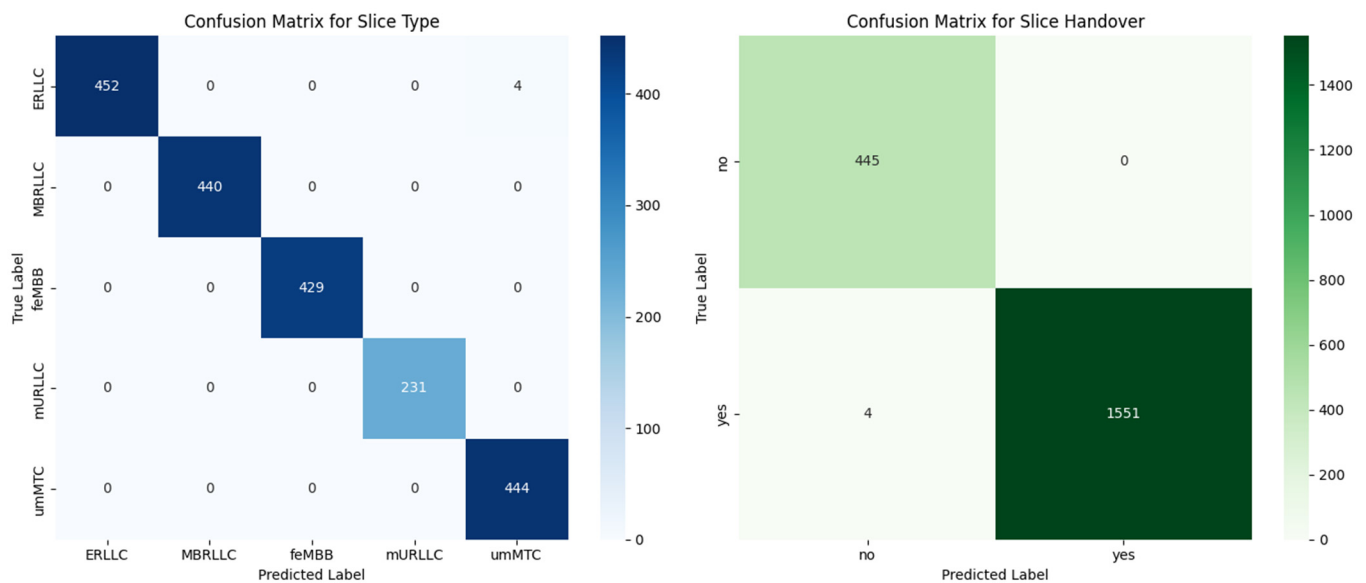


Figure 12. Confusion matrix for the SVM model validation set after hyperparameter tuning.

The number of incorrect predictions has decreased drastically, both in terms of slice type and handover decision, with only four cases of each incorrect prediction.

4.2. FNN Evaluation

Finally, we also evaluated the neural network that we trained on 30 epochs with a batch_size of 32. We calculated and plotted the loss and accuracy of our neural network, across all 30 epochs. The results are presented in Figures 13 and 14. The results are presented in Figures 13–15, and, as in the case of models based on RF classifiers or decision trees, respectively SVM, after tuning the hyperparameters we obtained an accuracy of over 99%.

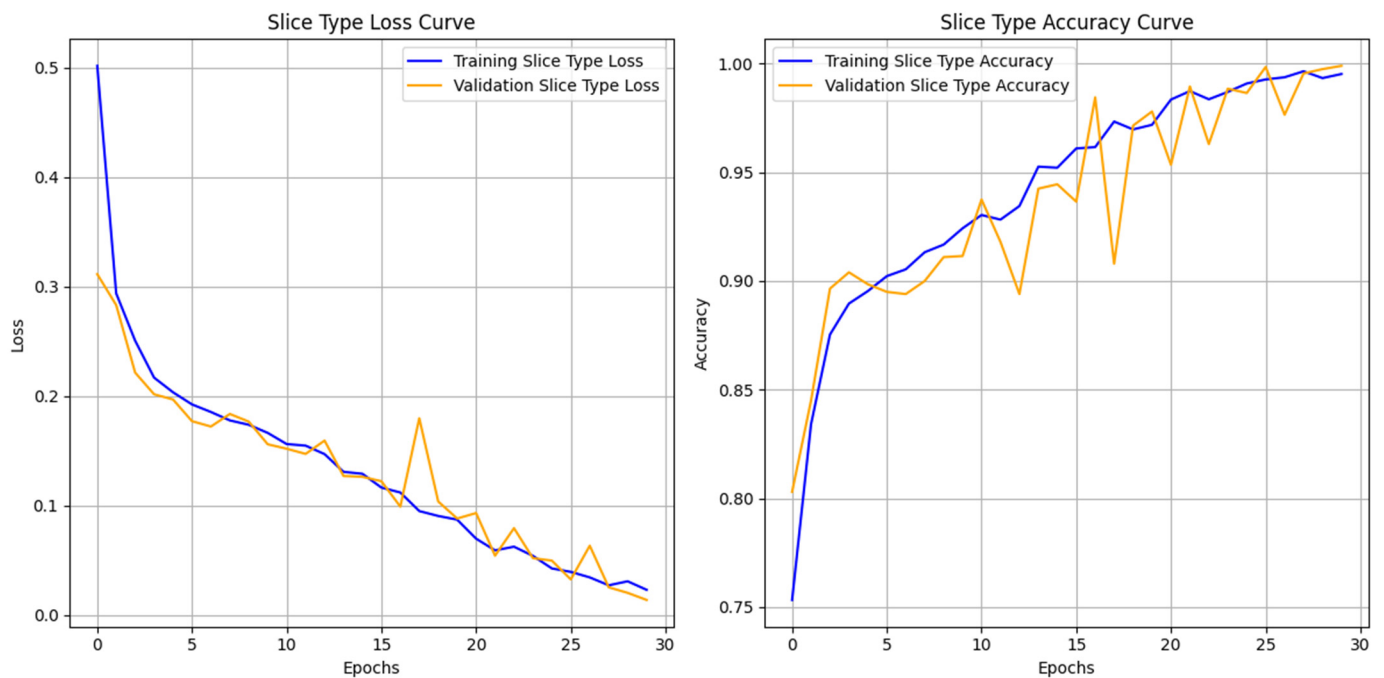


Figure 13. Loss and accuracy curve for slice type target.

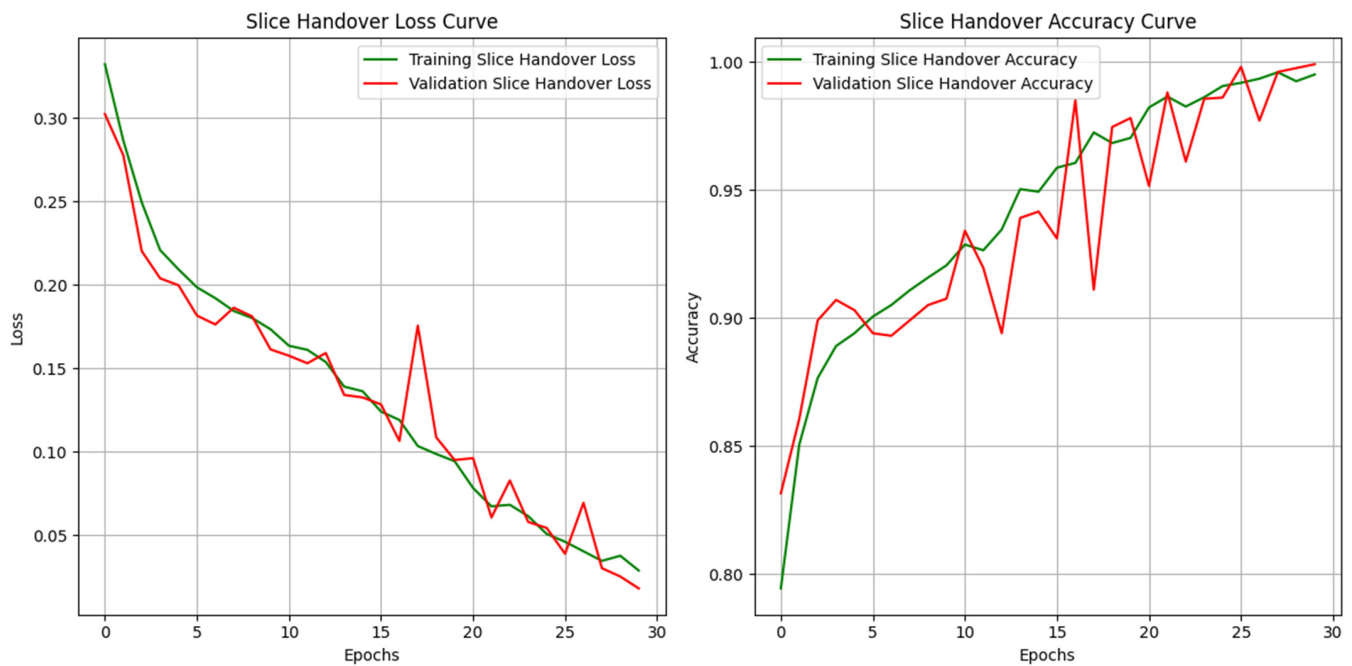


Figure 14. Loss and accuracy curve for slice handover target.

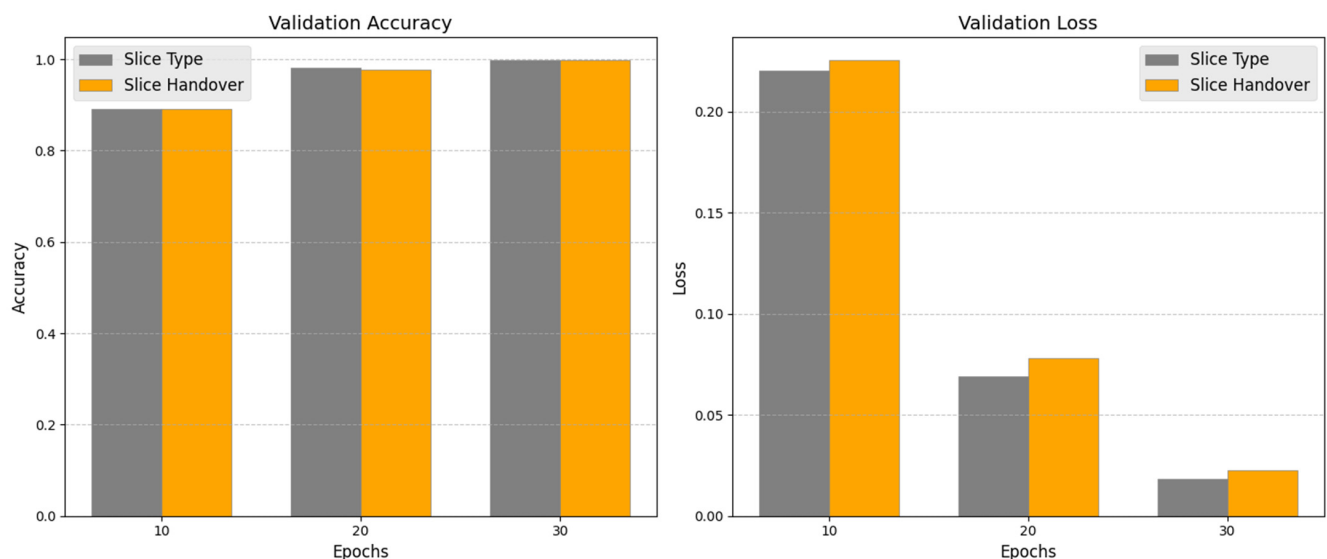


Figure 15. Comparison between the results obtained with a step of 10 in the number of epochs.

4.3. Scalability Evaluation

In our study we performed a scalability evaluation of machine learning algorithms with special emphasis on the Random Forest model by generating and testing five different dataset sizes (10,000, 20,000, 40,000, 60,000, and 100,000 samples). We analyzed the differences in training and prediction time while increasing the dataset size and examined the effect of parallelization on the performance of the model. To assess the scalability, experiments were conducted using different thread counts by varying the parameter `n_jobs` in the RF model. The `n_jobs` values under consideration were 1, 2, 4, 8, and -1 . We recorded training and prediction time for each setting for various sizes of datasets. This allowed us to observe the influence of the parallelism on the model's runtime, specifically focusing on how it affects both the training and prediction processes. The comparison is illustrated in Figure 16.

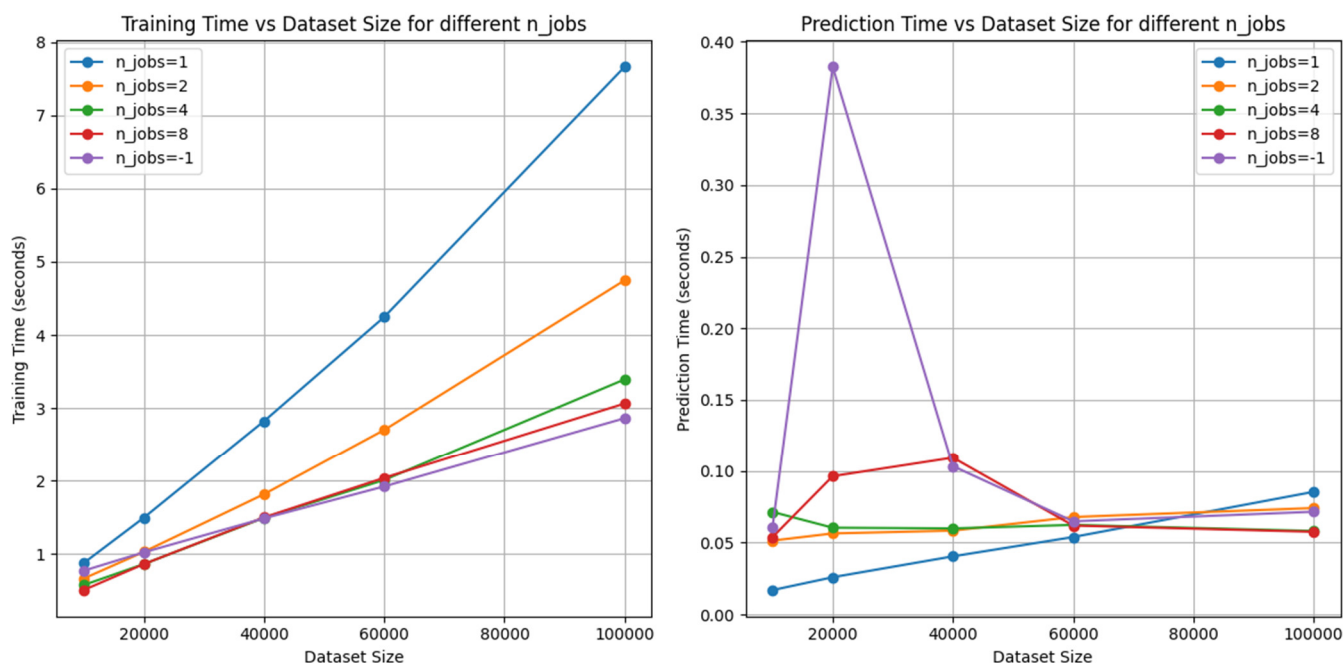


Figure 16. Training and prediction time vs. dataset size for varying parallelization configurations.

We begin by highlighting the results obtained with $n_jobs = 1$, where the training and prediction times increase gradually with the dataset size. For instance, when processing the 10,000-sample dataset, the training time was 0.8753 s, and the prediction time was 0.017 s. As the dataset increased to 100,000 samples, the training time rose to 7.6637 s, while the prediction time increased to 0.0853 s. This trend shows that as the dataset size increases, the computational demands of the model scale in a predictable way by going up. Next, we observed the results with $n_jobs = 2$ and found that parallelization led to reduced training times. For example, the training time for the 10,000-sample dataset decreased to 0.6653 s compared to 0.8753 s with $n_jobs = 1$, and similarly, the training time for the 100,000-sample dataset reduced from 7.6637 s to 4.7479 s. However, the prediction time showed less fluctuation with varying n_jobs values. For instance, while the prediction time of the 10,000-sample dataset was 0.017 s for $n_jobs = 1$, it increased to 0.0514 s for $n_jobs = 2$. This indicates that the computational demands for prediction are less sensitive to parallelization compared to training. Increasing the number of threads further by setting $n_jobs = 4$ and 8, we saw the trend continue downward in training time. The 100,000-sample dataset's training time went down from 4.7479 s with $n_jobs = 2$ to 3.3939 s with $n_jobs = 4$ and further to 3.0637 s with $n_jobs = 8$. The prediction time showed some variability, but it did not increase linearly with the increase in threads, indicating that the prediction step is not strongly affected by parallelism. Specifically, the prediction time for the dataset with 100,000 samples decreased only modestly from 0.0853 s when using $n_jobs = 1$ to 0.0575 s when using $n_jobs = 8$. A key observation from our experimental analysis is that there are diminishing returns in terms of the reduction in prediction time with an increase in the number of threads. This behavior is likely due to the inherent nature of the prediction phase in the RF model, which is generally less parallelizable than the training phase. The training phase involves building decision trees, which benefits significantly from parallel computation across multiple processors. In contrast, prediction involves aggregating results from individual trees, a process that is less computationally intensive and does not scale as efficiently with additional threads. In addition, we observed that the prediction time fluctuated in some cases when the number of threads was set to $n_jobs = -1$, which utilizes all available cores. For example, the prediction time for the

10,000-sample dataset with $n_{\text{jobs}} = -1$ was 0.0608 s, while for the 20,000-sample dataset, it increased substantially to 0.3825 s. This anomaly in prediction time could be attributed to the specific system architecture, resource contention, or other factors related to how the operating system manages task distribution across multiple CPU cores. Despite these fluctuations, the training time benefits from parallelization remained consistent, showing a reduction in training time as more threads were employed.

5. Conclusions

Network slicing remains a critical factor for the next generation of networks, 6G, through which stringent service and traffic requirements, resilience, scalability, or network security can be ensured. This paper introduced a synthetic dataset, obtained from a review of recent articles that introduce and present future 6G services and give clues regarding their traffic requirements. We trained the generated data using various machine learning models based on classifiers such as random forest, decision trees, kNN, Naïve Bayes, or SVM, but also through an FNN neural network. We optimized the models so that on most of them (random forest, decision trees, SVM, and FNN), we managed to obtain an accuracy of over 99% both in terms of correctly classifying the type of slice in 6G and in terms of deciding the handover operation on the current slice.

As a comparison with state-of-the-art publications, we did not find any dataset that strictly addresses future 6G services. Therefore, we believe that our proposed dataset, even if in an incipient state, may be useful in future research on the development of network slicing mechanisms for 6G assisted by artificial intelligence.

As future contributions, we want to improve the dataset to be as close to reality as possible. Obviously, this is not possible now, as 6G standardization is in progress. We plan to address this limitation by leveraging transfer learning techniques. We aspire to develop an artificial intelligence-assisted network slicing mechanism capable of performing the operations described in this work in real time, incorporating transfer learning and domain adaptation strategies to facilitate seamless deployment in future 6G environments. Furthermore, we plan to explore the integration of additional security-related features into the dataset, such as attack signatures or vulnerability profiles, to further strengthen the ability of AI models to address security concerns in 6G networks.

Author Contributions: Conceptualization, R.B. and D.Z.; software, R.B.; validation, R.B. and D.Z.; investigation, R.B. and D.Z.; data curation, R.B.; writing—original draft preparation, R.B.; writing—review and editing, R.B., D.Z. and V.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author, R.B., upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Thantharate, A.; Paropkari, R.; Walunj, V.; Beard, C. DeepSlice: A Deep Learning Approach towards an Efficient and Reliable Network Slicing in 5G Networks. In Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; pp. 0762–0767. [[CrossRef](#)]
2. Saad, W.; Bennis, M.; Chen, M. A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems. *IEEE Netw.* **2020**, *34*, 134–142. [[CrossRef](#)]
3. Rasti, M.; Taskou, S.K.; Tabassum, H.; Hossain, E. Evolution Toward 6G Multi-Band Wireless Networks: A Resource Management Perspective. *IEEE Wirel. Commun.* **2022**, *29*, 118–125. [[CrossRef](#)]

4. Hossain, A.R.; Liu, W.; Ansari, N.; Kiani, A.; Saboorian, T. AI-Native for 6G Core Network Configuration. *IEEE Netw. Lett.* **2023**, *5*, 255–259. [CrossRef]
5. Alwakeel, A.M.; Alnaim, A.K. Network Slicing in 6G: A Strategic Framework for IoT in Smart Cities. *Sensors* **2024**, *24*, 4254. [CrossRef] [PubMed]
6. Yang, P.; Xiao, Y.; Xiao, M.; Li, S. 6G Wireless Communications: Vision and Potential Techniques. *IEEE Netw.* **2019**, *33*, 70–75. [CrossRef]
7. Botez, R.; Pasca, A.-G.; Sferle, A.-T.; Ivanciu, I.-A.; Dobrota, V. Efficient Network Slicing with SDN and Heuristic Algorithm for Low Latency Services in 5G/B5G Networks. *Sensors* **2023**, *23*, 6053. [CrossRef]
8. Zhang, Z.; Xiao, Y.; Ma, Z.; Xiao, M.; Ding, Z.; Lei, X.; Karagiannidis, G.K.; Fan, P. 6G Wireless Networks: Vision, Requirements, Architecture, and Key Technologies. *IEEE Veh. Technol. Mag.* **2019**, *14*, 28–41. [CrossRef]
9. Banafaa, M.; Shaya, I.; Din, J.; Azmi, M.H.; Alashbi, A.; Daradkeh, Y.I.; Alhammadi, A. 6G Mobile Communication Technology: Requirements, Targets, Applications, Challenges, Advantages, and Opportunities. *Alex. Eng. J.* **2022**, *64*, 245–274. [CrossRef]
10. Alwis, C.D.; Kalla, A.; Pham, Q.V.; Kumar, P.; Dev, K.; Hwang, W.J.; Liyanage, M. Survey on 6G Frontiers: Trends, Applications, Requirements, Technologies and Future Research. *IEEE Open J. Commun. Soc.* **2021**, *2*, 836–886. [CrossRef]
11. Botez, R.; Costa-Requena, J.; Ivanciu, I.-A.; Strautiu, V.; Dobrota, V. SDN-Based Network Slicing Mechanism for a Scalable 4G/5G Core Network: A Kubernetes Approach. *Sensors* **2021**, *21*, 3773. [CrossRef] [PubMed]
12. 3rd Generation Partnership Project. Technical Specification Group Services and System Aspects. Management and Orchestration; Concepts, Use Cases and Requirements; (Release 17). 3GPP TS 28.530 V17.2.0. European Telecommunications Standards Institute. 2022. Available online: https://www.etsi.org/deliver/etsi_ts/128500_128599/128530/17.02.00_60/ts_128530v170200p.pdf (accessed on 7 January 2025).
13. 3rd Generation Partnership Project. Technical Specification Group Services and System Aspects. Study on Enhancement of Network Slicing Phase 2 (Release 17). 3GPP TR 23.700-40 V17.0.0. 2021. Available online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3687> (accessed on 7 January 2025).
14. 3rd Generation Partnership Project. Technical Specification Group Services and System Aspects. Study on Enhancement of Network Slicing Phase 3 (Release 18). 3GPP TR 23.700-41 V18.0.0. 2022. Available online: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=4002> (accessed on 7 January 2025).
15. Thantharate, A.; Beard, C.; Paropkari, R.; Walunj, V. CRAWDAD umkc/networkslicing5g; IEEE Dataport: New York, NY, USA, 2022. [CrossRef]
16. Rojas, J.S. IP Network Traffic Flows Labeled with 75 Apps—Labeled IP Flows with Their Application Protocol. Figshare. 2017. Available online: <https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps> (accessed on 8 December 2024).
17. 3GPP. TS 23.501 Version 16.6.0 Release 16: System Architecture for the 5G System (5GS). 2020. Available online: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf (accessed on 8 December 2024).
18. Liu, Y.; Deng, Y.; Nallanathan, A.; Yuan, J. Machine Learning for 6G Enhanced Ultra-Reliable and Low-Latency Services. *IEEE Wirel. Commun.* **2023**, *30*, 48–54. [CrossRef]
19. Abidi, M.H.; Alkhalefah, H.; Moiduddin, K.; Alazab, M.; Mohammed, M.K.; Ameen, M.K.; Gadekallu, T.R. Optimal 5G Network Slicing Using Machine Learning and Deep Learning Concepts. *Comput. Stand. Interfaces* **2021**, *76*, 103518. [CrossRef]
20. Mohammedali, N.A.; Kanakis, T.; Al-Sherbaz, A.; Agyeman, M.O. Traffic Classification Using Deep Learning Approach for End-to-End Slice Management in 5G/B5G. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; pp. 357–362. [CrossRef]
21. Gabilando, A.; Fernandez, Z.; Viola, R.; Martín, Á.; Zorrilla, M.; Angueira, P.; Montalbán, J. Traffic Classification for Network Slicing in Mobile Networks. *Electronics* **2022**, *11*, 1097. [CrossRef]
22. Bega, D.; Gramaglia, M.; Fiore, M.; Banchs, A.; Costa-Perez, X. DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 280–288. [CrossRef]
23. Domeke, A.; Cimoli, B.; Monroy, I.T. Integration of Network Slicing and Machine Learning into Edge Networks for Low-Latency Services in 5G and Beyond Systems. *Appl. Sci.* **2022**, *12*, 6617. [CrossRef]
24. Dangi, R.; Lalwani, P. Optimizing Network Slicing in 6G Networks through a Hybrid Deep Learning Strategy. *J. Supercomput.* **2024**, *80*, 20400–20420. [CrossRef]
25. 6G Network Slicing Dataset. 2025. Available online: <https://github.com/robertbotez/6g-network-slicing-dataset> (accessed on 7 December 2024).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.