DrillBit

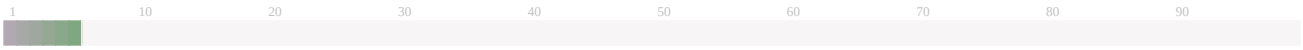
The Report is Generated by DrillBit Plagiarism Detection Software

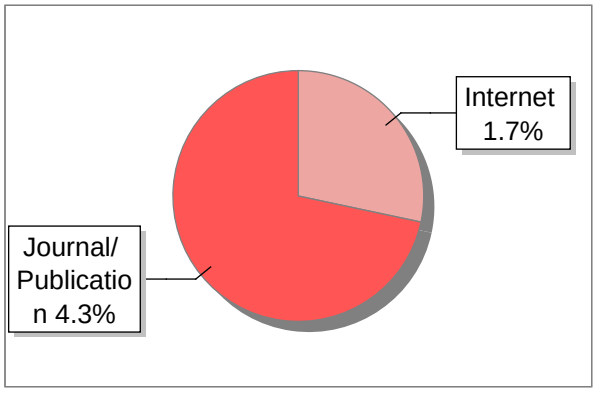
Submission Information

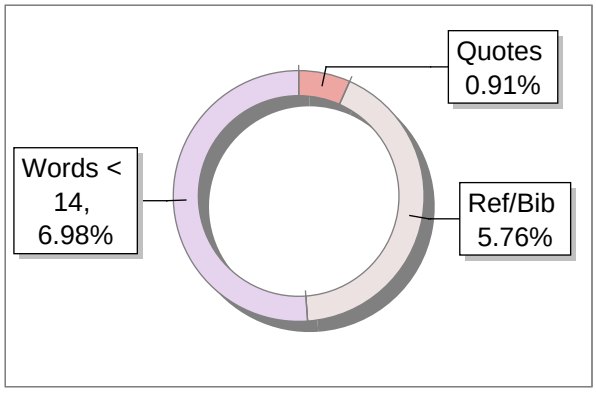
Author Name	Mohan Kumar
Title	100_Project Report_new
Paper/Submission ID	1145451
Submitted by	pallabikar@pes.edu
Submission Date	2023-11-30 08:29:43
Total Pages	63
Document type	Project Work

Result Information

Similarity **6 %**







Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Sources: Less than 14 Words Similarity	Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

6	20	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	Big data in cybersecurity a survey of applications and future trends by Alani-2021	1	Publication
2	Automation of 5G Network Slice Control Functions with Machine Learning by Kafle-2019	1	Publication
3	www.frontiersin.org	1	Publication
4	Computation Offloading and Retrieval for Vehicular Edge Computing Algorithms, M by Boukerche-2020	<1	Publication
5	www.mageplaza.com	<1	Internet Data
7	www.mdpi.com	<1	Internet Data
8	Regularized Semipaired Kernel CCA for Domain Adaptation by Mehrkanoon-2017	<1	Publication
9	www.dx.doi.org	<1	Publication
10	www.readbag.com	<1	Internet Data
12	docplayer.net	<1	Internet Data
13	Thesis Submitted to Shodhganga, shodhganga.inflibnet.ac.in	<1	Publication
14	Side Channel Attack-Aware Resource Allocation for URLLC and eMBB Slice by Li-2020	<1	Publication

15	A grammar inference approach for language self-adaptation and evolution in digit by Ferri-2019	<1	Publication
16	A multirobot target searching method based on bat algorithm in unknown by Tang-2019	<1	Publication
17	IEEE 214 IEEE 53rd Annual Conference on Decision and Control (CDC) by	<1	Publication
18	docplayer.net	<1	Internet Data
19	www.computermusicjournal.org	<1	Internet Data
20	abnews-wire.blogspot.com	<1	Internet Data
21	A loop-flow-based method for capacitor optimization in distribution sy by BM-2005	<1	Publication
23	repositorio.unal.edu.co	<1	Internet Data

INTRODUCTION

1.1 Network Slicing

Network slicing is a fundamental component of 5G networks that allows service providers to create virtual networks tailored to specific use cases. It enables the deployment of multiple logical networks on a common physical infrastructure, providing dedicated bandwidth, latency, and reliability for each use case.

Network slicing facilitates the provision of differentiated services with specific QoS requirements such as enhanced mobile broadband, massive machine type communications, and ultra-reliable low latency communications. It also optimizes resource utilization by sharing the physical infrastructure across different use cases.

To achieve network slicing, service providers use various slicing techniques, including software-defined networking (SDN), network function virtualization (NFV), and cloud computing. These techniques enable the creation, management, and orchestration of virtual networks.

Overall, network slicing is a crucial feature of 5G networks that enables service providers to offer differentiated services with specific QoS requirements, while also optimizing resource utilization. It has the potential to drive innovation and create new business opportunities in various industries, including healthcare, transportation, and manufacturing.

1.2 Dynamic Resource Allocation

Dynamic resource allocation is an essential feature of 5G networks that enables service providers to allocate network resources efficiently based on current traffic patterns and user requirements. It ensures that critical use cases receive the required resources in real-time, resulting in an enhanced user experience.

Dynamic resource allocation enables service providers to allocate resources such as bandwidth, processing power, and storage capacity dynamically, optimizing the network's overall performance. To achieve this, various techniques such as machine learning, predictive algorithms, and artificial intelligence are used to predict traffic patterns and allocate resources in real-time.

Overall, dynamic resource allocation plays a crucial role in ensuring that users receive the required QoS metrics, resulting in an enhanced user experience. It is expected to drive innovation and create new business opportunities in various industries, including healthcare, transportation, and manufacturing.

1.1.3 Quality Of Service Framework

A Quality of Service (QoS) framework is a set of guidelines and policies that govern the allocation of network resources, ensuring that users receive the required bandwidth, latency, and reliability for their specific use case. It is an essential component of 5G networks that enables service providers to prioritize traffic based on user requirements and optimize network performance.

The QoS framework provides mechanisms for monitoring and controlling network traffic to ensure that QoS metrics are met. Service providers use various techniques such as traffic shaping, packet scheduling, and admission control to allocate network resources efficiently and ensure that users receive the required QoS metrics.

Overall, a QoS framework plays a crucial role in enhancing the user experience and optimizing network performance. It is expected to drive innovation and create new business opportunities in various industries, including healthcare, transportation, and manufacturing.

PROBLEM STATEMENT

In the dynamic realm of 5G networks, where diverse applications demand exemplified Quality of Service (QoS), efficiently allocating network resources has become an intricate challenge. While network slicing holds the promise of customized virtual networks, the effective distribution and real-time management of resources among these slices necessitate an intelligent QoS framework. Issues such as congestion, interference, and suboptimal resource allocation often lead to compromised QoS, impacting user experience negatively. Traditional methods fall short in addressing these intricacies. Our project bridges this gap through an exhaustive comparative study encompassing traditional algorithms, supervised machine learning models, and cutting-edge reinforcement learning techniques.

Our primary goal is dual-fold:

1. Firstly, to empirically establish that supervised machine learning surpasses traditional and models in the context of 5G network slicing. Rigorous analysis substantiates its superior predictive accuracy and adaptability, especially in dynamic network conditions.
2. Secondly, our project endeavours to introduce a pragmatic and intelligent QoS framework. This framework not only sets a benchmark for network administrators but also equips them with real-time insights. By offering practical recommendations for optimal network slice selection tailored to specific applications, our QoS framework emerges as an indispensable tool for network administrators navigating the complexities of 5G networks. The project culminates in the development of an unyielding framework ensuring unwavering Quality of Service in 5G network slicing, addressing a critical need in modern network management.

CHAPTER 3

LITERATURE REVIEW

3.1 Resource Allocation in Network Slicing: Leveraging Deep Reinforcement Learning [1] (RESEARCH PAPER - 1)

3.1.1 Introduction:

A potential method called network slicing allows operators to divide a single physical network into numerous logical networks, each of which is optimised for a particular use case. The complex and dynamic nature of the network environment makes it difficult to manage resources effectively in network slicing.The authors of this work suggest a deep reinforcement learning (DRL) method.

3.1.2 Characteristics and Implementation:

The proposed DRL approach is based on a multi-agent system, where each agent represents a slice and is responsible for managing the resources of that slice. The agents use a deep neural network to learn the optimal resource allocation policy for their slice. The network input includes the current network state and the slice's resource demand, and the output is the resource allocation decision.

3.1.3 Features:

The DRL approach proposed in this paper has several key features. First, it is designed to handle the dynamic and complex nature of the network environment, allowing it to adapt to changes in network conditions. Second, it is able to learn from experience, enabling it to improve over time and optimize network performance. Third, it is a distributed approach that can be applied to large-scale networks.

3.1.4 Evaluation:

The authors evaluate the proposed DRL approach using a simulation environment based on a real-world network topology. The simulation results finalize that the DRL approach outperforms traditional resource management methods in terms of network performance metrics such as throughput, delay, and packet loss. The authors also perform a sensitivity analysis to evaluate the robustness of the DRL.

3.2 Synergetic Resource Allocation in Network Slicing: Enhanced Performance Through Distributed Bandwidth and Computational Allocation [5] (RESEARCH PAPER - 2.)

3.2.1 Introduction:

Network slicing is a promising technique for providing customized network services to different users or applications. However, efficient resource allocation is crucial for ensuring the quality of service (QoS) and maximizing the utilization of network resources. In this paper, the authors propose a distributed resource allocation approach for network slicing, which jointly allocates both bandwidth.

3.2.2 Characteristics and Implementation:

The proposed approach is based on a multi-agent system, where each agent represents a network slice and is responsible for managing the resources of that slice. The agents communicate with each other to coordinate resource allocation decisions and ensure that the overall network performance is optimized. The authors propose a distributed optimization algorithm that combines both primal-dual sub gradient methods and consensus-based algorithms to achieve efficient resource allocation. The proposed approach also takes into account the resource heterogeneity and dynamic network conditions, such as changing user demands.

3.2.3 Features:

The proposed approach has several key features. First, it is a distributed approach that allows for efficient and scalable resource allocation in large-scale network slicing scenarios. Second, it jointly considers both bandwidth and computational resources, which are critical for many emerging applications. Third, it takes into account the dynamic nature of the network environment and can adapt to changes in user demands and network conditions. Finally, the authors demonstrate through simulations that the proposed approach can achieve better network performance compared to traditional resource allocation methods.

3.2.4 Evaluation:

The authors evaluate the proposed approach through simulations based on a real-world network topology. The simulation results displays that the proposed approach can achieve better network performance in terms of resource utilization and QoS compared to traditional resource allocation methods.The suggested approach is also subjected to a sensitivity analysis by the authors, who demonstrate its adaptability to changing user needs and network conditions as well as its robustness under various network settings.Overall, the outcomes show how well the suggested method for distributing resource allocation in network slicing works.

3.3 Revolutionizing Networking Landscapes: Unveiling the Potential of Network Slicing in Customized Service Delivery and Resource Management [6] (RESEARCH PAPER - 3)

3.3.1 Introduction:

A promising idea in networking is network slicing, which enables the division of a single physical network infrastructure into several virtual networks, each of which is optimised for a particular set of services or applications.This enables network operators to offer differentiated services to their customers, with each slice providing a unique set of capabilities, performance guarantees, and security features. The implementation of network slicing involves several challenges related to the design of the underlying architecture, orchestration of resources, and management of network slices.

3.3.2 Characteristics and Implementation:

Network slicing is characterized by several key features, including the ability to support multiple virtual networks on a single physical infrastructure, the ability to dynamically allocate and manage resources across network slices, and the ability to isolate network slices from each other to ensure security and privacy. In order to construct virtual networks that may be customised to the unique requirements of various applications and services, network slicing is implemented using virtualization technologies, such as Software-defined Networking and Network Function Virtualization.

3.3.3 Features:

The features of network slicing can be broadly categorized into three categories: resource isolation, service differentiation, and dynamic resource management. Resource isolation involves the ability to create isolated virtual networks that are dedicated to specific applications or services, with each network having its own set of resources and performance guarantees. Service differentiation involves the ability to provide different levels of service quality, security, and reliability to different applications or services. Dynamic resource management involves the ability to allocate and manage resources across network slices in real-time, based on the changing needs of applications and services.

3.3.4 Evaluation:

The evaluation of network slicing involves several metrics, including network efficiency, scalability, flexibility, and security. Network efficiency is measured by the ability to maximize the use of available resources, while maintaining a high level of performance and service quality. Scalability is measured by the ability to support a large number of network slices, each with different requirements and characteristics. Flexibility is measured by the ability to adapt to changing network conditions and requirements, while maintaining a high level of service quality. Security is measured by the ability to ensure the isolation and privacy of network slices, and to prevent unauthorized access and attacks.

3.4 Real-time Resource Orchestration for Tailored Network Services: Implementing Dynamic Allocation in Network Slicing [4] (RESEARCH PAPER – 4)

3.4.1 Introduction:

Network slicing is a key enabler for 5G and beyond networks, as it allows network operators to offer differentiated services to their customers. Dynamic virtual resource allocation is a technique used in network slicing that enables resources to be allocated and de-allocated to network slices in real-time, based on the changing needs of applications and services. This paper discusses the implementation of dynamic virtual resource allocation in network slicing, and the challenges and opportunities it presents.

3.4.2 Characteristics and Implementation:

Dynamic virtual resource allocation involves the use of Software-Defined Networking and Network Function Virtualization technologies to born virtual networks that can be tailored to the relevant requirements of distinct applications and services. This enables resources to be allocated and de-allocated in real-time, based on the changing needs of applications and services. The implementation of dynamic virtual resource allocation involves the deployment of a centralized controller that manages the allocation of resources to network slices.

3.4.3 Features:

The features of dynamic virtual resource allocation include the ability to allocate resources in real-time, based on the changing needs of applications and services, the ability to optimize resource utilization across network slices, and the ability to provide different levels of service quality, security, and reliability to different applications and services. Dynamic virtual resource allocation also enables network operators to offer flexible pricing models, as resources can be allocated and de-allocated on a per-usage basis.

3.4.4 Evaluation:

The evaluation of dynamic virtual resource allocation involves several metrics, including resource utilization, service quality, scalability, and cost-effectiveness. Resource utilization is measured by the ability to maximize the use of available resources, while maintaining a high level of performance and service quality. Service quality is measured by the ability to provide different levels of service quality, security, and reliability to different applications and services. Scalability is measured by the ability to support a large number of network slices, each with different requirements and characteristics. Cost-effectiveness is measured by the ability to optimize resource utilization and minimize costs, while maintaining a high level of service quality.

3.5 Optimizing End-to-End Network Slicing: Multi-Agent Reinforcement Learning for Resource Allocation and Adaptability [2] (RESEARCH PAPER - 5)

3.5.1 Introduction:

End-to-end network slicing is a key feature of 5G and beyond networks that enables network operators to offer differentiated services to their customers. Resource management is a cruical aspect of end-to-end network slicing, as it involves the allocation of resources to network slices to ensure optimal performance and service quality. This paper proposes a multi-agent reinforcement learning based approach to resource management in end-to-end network slicing, and discusses the challenges and opportunities it presents.

3.5.2 Characteristics and Implementation:

The proposed approach involves the use of multi-agent reinforcement learning (MARL) to manage resources in end-to-end network slicing. MARL involves the use of multiple agents, each responsible for managing a different aspect of the network slice, to optimize resource allocation and performance. The implementation of the approach involves the deployment of a centralized controller that manages the agents and their interactions, and the use of a reward function to incentivize agents to optimize resource allocation and performance.

3.5.3 Features:

The features of the proposed MARL-based approach include the ability to optimize resource allocation and performance in end-to-end network slicing, the ability to adapt to changing network conditions and requirements, and the ability to provide different levels of service quality, security, and reliability to different applications and services. The approach also enables network operators to offer flexible pricing models, as resources can be allocated and de-allocated on a per-usage basis.

3.5.4 Evaluation:

The evaluation of the MARL-based approach involves several metrics, including resource utilization, service quality, scalability, and convergence time. Resource utilization is measured by the ability to maximize the use of available resources, while maintaining a high level of performance and service quality. Service quality is measured by the ability to provide different levels of service quality, security, and reliability to different applications and services. Scalability is measured by the ability to support a large number of network slices, each with different requirements and characteristics. Convergence time is measured by the time required for the agents to converge to an optimal resource allocation and performance.

3.6 Secure5G: Deep Learning-Powered Framework for Safeguarding Network Slicing Security [7] (RESEARCH PAPER - 6)

3.6.1 Introduction:

Network slicing is a key feature of 5G and beyond networks that enables network operators to offer differentiated services to their customers. However, network slicing also introduces security challenges, as different network slices may have different security requirements and vulnerabilities. This paper proposes a deep learning-based framework, called Secure5G, for secure network slicing in 5G and beyond networks, and discusses the challenges and opportunities it presents.

3.6.2 Characteristics and Implementation:

The Secure5G framework involves the use of deep learning techniques to detect and mitigate security threats in network slices. The framework includes a security module that analyzes network traffic and identifies potential threats, and a response module that takes appropriate actions to mitigate the threats. The implementation of the framework involves the deployment of a centralized controller that manages the security module and the response module, and the use of a feedback loop to optimize the performance and security of the network slices.

3.6.3 Features:

The features of the Secure5G framework include the ability to detect and mitigate security threats in real-time, the ability to adapt to changing network conditions and requirements, and the ability to provide different levels of security and privacy to different applications and services. The framework also enables network operators to offer flexible pricing models, as resources can be allocated and de-allocated based on security requirements.

3.6.4 Evaluation:

The evaluation of the Secure5G framework involves several metrics, including security effectiveness, resource utilization, service quality, and scalability. Security effectiveness is measured by the ability of the framework to detect and mitigate security threats in network slices. Resource utilization is measured by the ability of the framework to optimize resource allocation while maintaining a high level of security and service quality. Service quality is measured by the ability of the framework to provide different levels of service quality and privacy to different applications and services. Scalability is measured by the ability of the framework to support a large number of network slices, each with different security requirements and characteristics.

3.7 Deep Learning and Blockchain Integration for Enhanced Security in 5G-Enabled IoT Networks [8] (RESEARCH PAPER - 7)

3.7.1 Introduction:

This paper proposes a security framework for intelligent 5G-enabled IoT that combines deep learning and blockchain technologies. The framework aims to address the security challenges faced by IoT devices in 5G networks, including attacks on device identity, data privacy, and network availability. The framework uses deep learning to detect and mitigate security threats, and blockchain to provide a secure and decentralized platform for data storage and management.

3.7.2 Characteristics and Implementation:

The proposed framework involves the use of deep learning algorithms to detect and classify security threats in IoT devices and networks, and to generate alerts and responses in real-time. The framework also uses blockchain to create a secure and transparent platform for data management, authentication, and access control. The implementation of the framework involves the deployment of edge nodes and gateways that can communicate with IoT devices, and a centralized or decentralized blockchain network that can provide secure and reliable data storage and management.

3.7.3 Features:

The features of the proposed framework include the ability to detect and mitigate security threats in real-time, the ability to provide secure and decentralized data storage and management, the ability to support different levels of access control and privacy, and the ability to integrate with existing 5G network infrastructures. The framework also enables network operators to offer flexible pricing models, as resources can be allocated and de-allocated based on security and performance requirements.

3.7.4 Evaluation:

The evaluation of the proposed framework involves several metrics, including security effectiveness, resource utilization, service quality, and scalability. Security effectiveness is measured by the ability of the framework to detect and mitigate security threats in IoT devices and networks. Resource utilization is measured by the ability of the framework to optimize resource allocation while maintaining a high level of security and service quality. The framework's capacity to offer various levels of service quality and privacy to various IoT devices and apps serves as a gauge of service quality. The framework's

capacity to accommodate a vast number of IoT devices and apps, each with a variety of security requirements and features, is a key indicator of its scalability.

3.8 Machine Learning-Driven Automation for 5G Network Slice Control Functions [10] (RESEARCH PAPER - 8)

3.8.1 Introduction:

This paper proposes an automated approach to control the functions of 5G network slices using machine learning techniques. The goal is to enable faster and more efficient resource allocation and optimization for network slices. The proposed approach uses reinforcement learning to optimize the network slice functions, including the allocation of radio resources, computation resources, and storage resources.

3.8.2 Characteristics and Implementation:

The proposed approach involves the use of reinforcement learning algorithms to learn the optimal control policies for network slice functions. The algorithms use real-time data from the network to optimize the allocation of resources based on the performance requirements of the network slices. The implementation of the approach involves the deployment of a learning agent that interacts with the network and learns the optimal control policies over time. The agent can be deployed centrally or in a distributed way across the network, depending on the network topology and requirements.

3.8.3 Features:

The features of the proposed approach include the ability to optimize the allocation of resources in real-time, the ability to learn and adapt to changing network conditions and requirements, and the ability to provide efficient resource allocation and optimization for different types of network slices. The approach also enables network operators to offer flexible pricing models based on the performance and resource requirements of the network slices.

3.8.4 Evaluation:

The evaluation of the proposed approach involves several metrics, including the performance of the network slices, the efficiency of resource allocation, and the scalability of the approach. The approach's capacity to satisfy the performance criteria of the network slices serves as a proxy for slices. The capacity of the strategy to distribute resources optimally while minimising resource waste serves as a gauge of resource allocation efficiency. The approach's scalability is determined by its capacity to manage a large number of network slices and adjust to shifting network circumstances and requirements.

3.9 Enhancing 5G Radio Access Network Slicing Through Intelligent Resource Scheduling [3] (RESEARCH PAPER - 9)

3.9.1 Introduction:

For 5G radio access network (RAN) slicing, this study suggests an intelligent resource scheduling method that attempts to maximise the use of radio resources and enhance overall system performance. The suggested method makes use of machine learning techniques to forecast network slice traffic demands and dynamically assign radio resources to satisfy various network slice performance criteria.

3.9.2 Characteristics and Implementation:

The proposed approach involves two main stages: traffic demand prediction and resource allocation. In the traffic demand prediction stage, a machine learning model is trained to predict the traffic demand of network slices based on historical data and real-time network measurements. The anticipated traffic demand is utilised to dynamically distribute radio resources to various network slices in order to satisfy their performance needs. The method's implementation entails the installation of a centralised controller that communicates with the RAN nodes and distributes resources in accordance with the forecasts and performance specifications of network slices.

3.9.3 Features:

The features of the proposed approach include the ability to predict traffic demand and allocate resources dynamically, the ability to adapt to changing network conditions and traffic patterns, and the ability to optimize the utilization of radio resources. The approach also enables network operators to offer flexible service-level agreements (SLAs) to different network slices based on their performance requirements.

3.9.4 Evaluation:

The evaluation of the proposed approach involves several metrics, including the resource utilization, network throughput, and the performance of network slices. The results show that the proposed approach outperforms traditional resource allocation schemes in terms of resource utilization and network throughput while meeting the performance requirements of different network slices. The approach also demonstrates the ability to adapt to changing traffic patterns and optimize the utilization of radio resources.

3.10 Exploring the Landscape of 5G Network Slicing: Challenges, Opportunities, and Use Cases [11] (RESEARCH PAPER - 10)

3.10.1 Introduction:

The challenges and opportunities that network slicing in 5G networks presents are discussed in this study. Enhanced mobile broadband (eMBB), ultra-reliable and low-latency communications (URLLC), and massive machine-type communications (mMTC) are just a few of the use cases that are covered in this article. It also covers the necessity for network slicing, its advantages, and the several use cases it can enable.

3.10.2 Characteristics and Implementation:

The ability to divide the network into various virtual networks, each with its own resources and administration tools, is one of the features of network slicing that are covered in the article. It also

discusses the different ways that network slicing can be implemented, such as through the use of technologies like software-defined networking (SDN) and network function virtualization (NFV).

3.10.3 Features:

The features of network slicing include the ability to provide customized services to different users or applications, the ability to optimize resource utilization, and the ability to enable flexible service-level agreements (SLAs) between network operators and their customers. Network slicing also allows for dynamic resource allocation, improved network efficiency, and reduced operational costs.

3.10.4 Evaluation:

The paper discusses the challenges and opportunities associated with network slicing, including the need for standardized interfaces, orchestration, and management frameworks. It also highlights the importance of security, privacy, and trust in network slicing, as well as the need for efficient resource allocation algorithms and effective service-level agreements. The paper concludes by discussing the potential impact of network slicing on various industries, such as healthcare, transportation, and manufacturing.

CHAPTER 4

PROJECT REQUIREMENTS SPECIFICATION

4.1 System Features

The Machine Intelligence Based Quality of Service (QoS) Framework is designed to optimize resource allocation and enable the creation and management of network slices with varying QoS requirements. The major features of the product include:

- **Resource allocation optimization**: The system leverages machine intelligence algorithms to optimize resource allocation in real-time. This ensures that network resources are

allocated efficiently and effectively, thereby reducing network congestion thus, improving network performance.

- **QoS management**: The system enables the creation and management of network slices with varying QoS requirements. This allows network administrators to configure network slices according to specific traffic management policies and prioritize traffic based on the importance of the data.

- **Real-time monitoring:** The system provides real-time monitoring and management of network slices, enabling network administrators to detect and address issues as they arise. This ensures that network performance remains consistent and any issues can be resolved quickly.

- **User management**: The system enables network administrators to manage users and assign them to specific network slices based on their requirements. This ensures that users receive the appropriate level of service and that network resources are allocated optimally.

- **Scalability:** The system is designed to be scalable and can be deployed on a cloud-based server. This allows network administrators to manage their networks on a larger scale and respond quickly to changing network demands.

- **Flexibility:** The system is designed to be flexible and customizable, enabling network administrators to configure network slices according to their specific requirements. This allows for greater control over network resources.

4.2 Operating Environment

The Machine Intelligence Based Quality of Service (QoS) Framework can be deployed on a variety of hardware platforms and operating systems. The system's software components can be installed and configured on a cloud-based server, which can be accessed remotely by network administrators and service providers. The following are the operating environment requirements for the system:

- **Hardware platform**: The system can be deployed on a variety of hardware platforms, including servers, workstations, and laptops. The hardware platform should meet the minimum requirements for running the system's software components, including CPU, RAM, and storage space.

- **Operating system**: The system's software components are compatible with multiple operating systems, including Windows, Linux, and macOS. The operating system version should be up-to-date and compatible with the software components.

- **Software components**: The system's software components include machine intelligence algorithms, network slicing modules, real-time monitoring and management tools, and user management tools. These components should be installed and configured according to the system's documentation and requirements.

- **Network infrastructure**: The system requires a stable and reliable network infrastructure to function properly. The network infrastructure should be capable of handling the data traffic generated by the system and should have sufficient bandwidth and capacity to support the system's requirements.

It is designed to operate in a flexible and scalable environment. The system's software components can be installed and configured on a variety of hardware platforms and operating systems, enabling users to deploy the system according to their specific requirements. The system's network infrastructure requirements are minimal, and the system can be accessed remotely, making it a versatile and efficient solution for network administrators and service providers.

4.3 General Constraints, Assumptions and Dependencies

There are several constraints, assumptions, and dependencies that developers of the Machine Intelligence Based Quality of Service (QoS) Framework should be aware of. These include:

- **Regulatory policies:** The system should comply with all relevant regulatory policies, such as data protection and privacy laws. This may limit the choices available to developers in terms of the system's design and implementation.

- **Hardware limitations**: The system's performance may be affected by hardware limitations, such as signal timing requirements or insufficient processing power. Developers may need to optimize the system's algorithms and architecture to accommodate these limitations.

- **Limitations of simulation programs**: The accuracy and reliability of the system's simulation programs may be limited by factors such as the quality and availability of data. Developers may need to incorporate additional data sources or improve data processing methods to improve the accuracy of the simulations.

- **Interfaces to other applications**: The system may need to interface with other applications, such as network management tools or billing systems. Developers may need to ensure that the system's interfaces are compatible with these applications and that data can be transferred seamlessly between systems.

- **Parallel operations**: The system may need to perform parallel operations, such as managing multiple network slices simultaneously. Developers may need to optimize the system's architecture and algorithms to ensure efficient parallel processing.

- **Criticality of application:** The system's performance may be critical for certain applications, such as emergency services or healthcare. Developers may need to prioritize the system's performance and reliability for these critical applications.

- **Safety and security considerations**: The system may handle sensitive data or operate in environments with strict safety or security requirements. Developers may need to implement additional security measures, such as encryption or access controls, to protect sensitive data and ensure system safety.

4.4 Risks

During the development of the Machine Intelligence Based Quality of Service (QoS) Framework, there are several risks that should be considered. These risks include:

- **Resource requirements**: The system may require significant computing resources, such as processing power and storage space, to function effectively. Failure to allocate sufficient resources may result in system performance issues and reduce the effectiveness of the QoS framework.

- **Technical complexity:** The QoS framework is a technically complex system that requires expertise in machine learning, networking, and real-time monitoring. Failure to manage technical complexity may result in delays, cost overruns, and reduced functionality.

- **Integration with existing systems**: The QoS framework may need to integrate with existing network infrastructure, including legacy systems and third-party tools. Integration may be challenging, and failure to integrate effectively may result in compatibility issues, reduced functionality, and increased costs.

- **Security risks:** The QoS framework may be vulnerable to security risks, such as hacking, data breaches, and denial-of-service attacks. Developers must ensure that the system is secure and follows best practices for data protection and security.

- **User adoption:** The QoS framework may be complex and require training for users. Failure to ensure user adoption may result in low usage rates, reduced functionality, and reduced return on investment.

- **Regulatory compliance**: The QoS framework may be subject to regulatory compliance requirements, such as data protection and privacy laws. Failure to comply with regulations may result in legal consequences and damage to the reputation of the organization.

These risks must be identified, evaluated, and addressed throughout the development process of the Machine Intelligence Based QoS Framework. Failure to manage these risks effectively may result in delays, cost overruns, and reduced functionality.

4.5 Functional Requirements

- **Slice creation:** The system should allow for the creation of network slices with specific requirements such as latency, throughput, and bandwidth.
- **Resource allocation:** The system should allocate the necessary resources such as compute, storage, and network resources to each slice based on their individual requirements.
- **Slice isolation:** The system should ensure that each slice is isolated from other slices to prevent interference and maintain security.
- **Dynamic scaling:** The system should be able to dynamically adjust the resources allocated to each slice based on changing requirements or demand.
- **Quality of service (QoS) management**: The system should prioritize traffic based on QoS requirements and ensure that each slice meets its performance objectives.
- **Fault tolerance:** The system should be resilient to faults and failures, and should be able to detect and recover from any issues that may arise.
- **Monitoring and analytics:** The system should provide real-time monitoring and analytics to enable operators to identify issues, optimize performance, and troubleshoot problems.

ii)Error handling and recovery:

The error handling and recovery mechanisms are incoperated so that the system can continue to operate even in the event of a failure or error which includes automated failover, backup and restore procedures, and proactive monitoring and alerting.

iii)Consequences of parameters:

The QoS framework must consider the consequences of all parameters used in the system, such as network latency, bandwidth, and packet loss. This includes analyzing the impact of changing these parameters and adjusting them as necessary to optimize

4.6 Hardware Requirements

The system should be able to run on a variety of hardware platforms, including desktops, laptops, servers, and mobile devices. The hardware components of the system should be connected through a standard communication protocol, such as Ethernet or Wi-Fi, and should be compatible with the operating system that the software product is running on.

The hardware components of the system should also meet certain minimum specifications in terms of processing power, memory, and storage capacity. The following are the minimum hardware requirements for the system:

- Processor: The system should be able to run on a processor with a clock speed of at least 2 GHz.
- Memory: The system should have a minimum of 8 GB of RAM to ensure that it can process large amounts of data quickly.
- Storage: The system should have a minimum of 100 GB of storage space to store the software product, as well as any data or files that are generated by the system.
- Network interface: The system should be connected to a network through a standard communication protocol, such as Ethernet or Wi-Fi, to allow for communication with other devices on the network.
- Operating system: The system should be compatible with the operating system that the software product is running on, such as Windows or Linux.

Overall, the hardware requirements for the system should be designed to ensure that the software product can run efficiently and effectively on a variety of hardware platforms and can process large amounts of data quickly.

4.7 Software Requirements

Name and Description: The software product that will be used in this project is a machine learning framework, which will be used to train and deploy machine learning models.

Version/Release Number: The latest stable version of the software product will be used in the project.

Operating Systems: The software product should be compatible with a variety of operating systems, including Windows, Linux, and MacOS.

Tools and Libraries: Python,Tensorflow,Scikit-learn,Pandas,Numpy,Matplotlib.

4.9 Performance Requirement

The performance requirements for the Machine Intelligence Based Quality of Service (QoS) Framework are as follows:

- Reliability: The system must be reliable and provide consistent results under normal operating conditions. The system should be able to handle a large number of requests without crashing or failing.

- Robustness: The system will be able to handle unexpected events and continue to function as expected. It should be able to recover from errors and continue to provide service.

- Availability: The system will be available for use at all times, except during scheduled maintenance periods. The system should be able to handle high traffic volumes without slowing down or crashing.

- Response Time: The system will respond quickly to user requests and provide results within a reasonable time frame. The response time should be consistent, regardless of the number of users or the complexity of the request.

- Scalability: The system will be scalable to handle increased traffic volumes and additional users. It should be able to handle increasing workloads without degradation in performance.

- Security: The system must be secure and protect user data and system resources from unauthorized access or malicious attacks.

CHAPTER 5

SYSTEM DESIGN

5.1 Current System

As per the proposed approach for resource allocation in network slicing, there is no specific current system being enhanced, replaced, or upgraded. However, if we consider the current approaches for resource allocation in network slicing, they mostly rely on heuristic-based methods or rule-based systems, which have limited capabilities to adapt to changing network conditions and lack the ability to learn from past experiences. These approaches also require a significant amount of manual intervention and do not take advantage of the vast amount of data available in the network. Hence, the proposed approach aims to address these limitations by leveraging machine learning techniques such as supervised and reinforcement learning.

5.2 Design Goals

1. Improved Resource Allocation: The primary goal of the proposed approach is to achieve improved resource allocation in network slicing, leading to better performance and efficient utilization of resources.
2. Scalability and Adaptability: The proposed approach should be scalable and adaptable to different network topologies and varying workload conditions. It should be able to adjust to changes in the network environment and the requirements of different applications and devices.
3. Performance: The proposed approach should be designed to deliver high performance in terms of throughput, latency, and packet loss. It should be able to handle high traffic loads and ensure fast and reliable service delivery
4. Security and Privacy: The proposed approach should ensure the security and privacy of user data and prevent unauthorized access and malicious attacks. It should also comply with privacy regulations and protect user privacy.
5. User Experience: The proposed approach should provide a good user experience, with intuitive interfaces and easy-to-use features. It should also ensure a consistent look and feel across different devices and platforms
6. Cost-effectiveness: The proposed approach should be cost-effective, with minimal hardware and software requirements and low operational costs. It should also provide a good return on investment and deliver business value.

5.3 Architecture Choices

- In designing the proposed solution for the resource allocation in network slicing, various architecture choices were considered. These choices included supervised learning, deep reinforcement learning, and hybrid approaches combining both techniques.
- The supervised learning approach was initially considered as it is a well-established method in machine learning, and there is a large body of literature on its application in resource allocation problems. However, one of the drawbacks of the supervised learning approach is that it requires a large amount of labeled data, which can be costly and time-consuming to obtain.
- On the other hand, deep reinforcement learning (DRL) has shown promising results in solving complex decision-making problems, such as resource allocation in network slicing. DRL can learn to optimize the allocation of resources based on the feedback from the environment. One

of the benefits of DRL is that it can learn from both labeled and unlabeled data, which can help address the issue of limited labeled data.

- However, the drawback of DRL is that it can be computationally expensive and requires a significant amount of training time. Moreover, the performance of DRL can be highly dependent on the choice of hyperparameters, and tuning them can be a challenging task.
- To overcome the limitations of both supervised learning and DRL, a hybrid approach was considered. In this approach, we can use supervised learning to pretrain a deep neural network on a large dataset and then fine-tune the network using DRL on a smaller labeled dataset. This can help leverage the benefits of both approaches and improve the performance of the model.
- In summary, while the supervised learning approach is well-established and easy to implement, it has limitations in terms of the requirement of large labeled data. DRL can overcome this limitation but can be computationally expensive and highly dependent on the choice of hyperparameters. The hybrid approach can combine the benefits of both approaches and can potentially lead to better performance.

5.4 High Level System Design

5.4.1 Design Description

We are first explaining how our machine learning model plays its role in the process of a user being allocated the right network slice for their requirement. So, the major modules at a high level involved in this process are:

- **The Network Administrator**: This entity is responsible for allocation of resources on the network slice and creates the right slice for the user.
- **The Service Provider**: This module is the connecting link between the user application and network admin. The user can only describe about their application and what are the necessary components so the service provider helps deliver the correct requirements to the machine intelligence aid with the help of which the network administrator understands the right slice to be assigned for the user.
- **Machine Intelligence**: The most efficient algorithm is already fed into this entity which receives the application of the user as the input and verifies if the slice being assigned is optimal and sufficient for the user or not. This gives a close-to-clear indication to the

network administrator about the network slice to be assigned based on the resources required for the proper and seamless functioning of the application.

5.5.2 5G Network Slicing Architectural Diagram

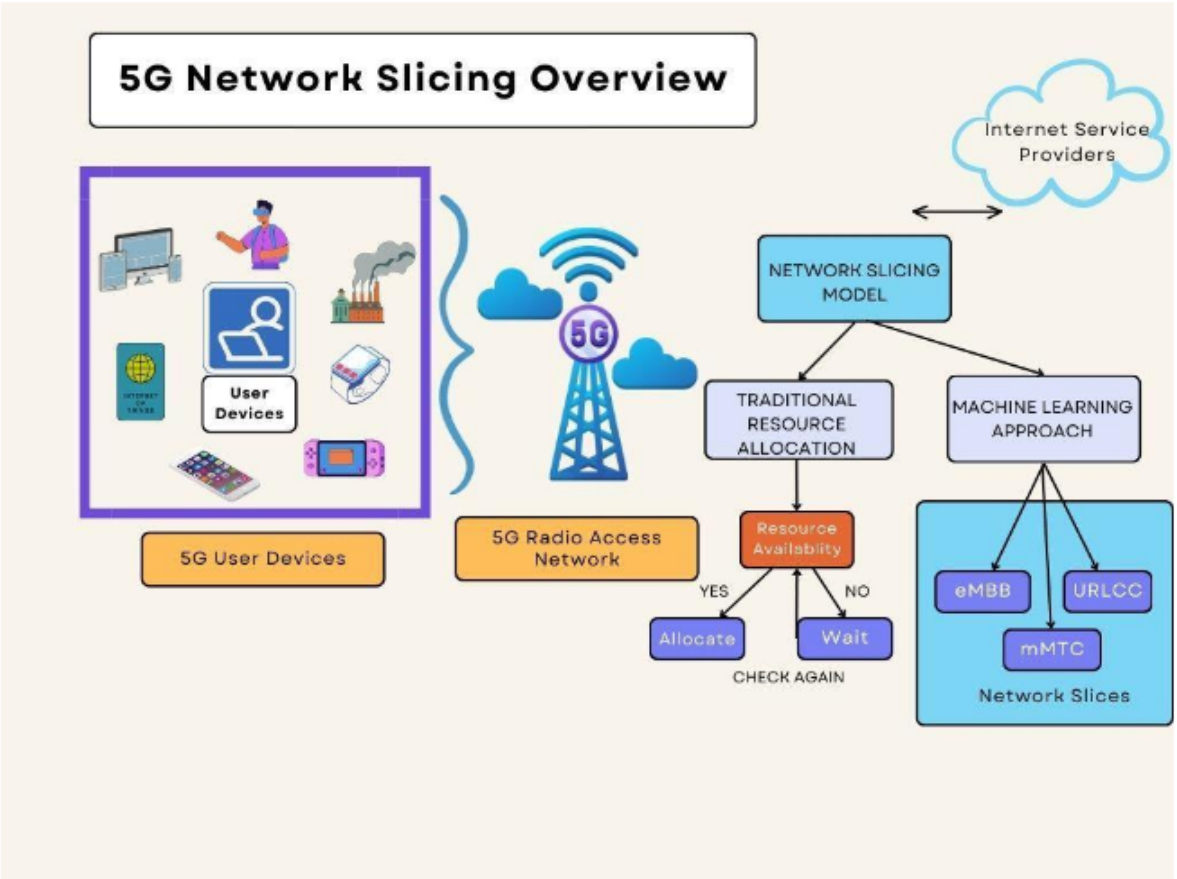


Fig-1 : 5G Network Slicing Architectural diagram

NetworkAdmin and **ServiceProvider** classes are responsible for managing the network and its resources. The **ServiceProvider** class has methods for creating network slices, managing network QoS, monitoring network usage, and managing user accounts. The **NetworkAdminclass** has methods for monitoring network usage, managing network slices, allocating network resources, and managing user accounts.

The **MachineIntelligence** class is responsible for analyzing network data, predicting network performance, optimizing resource allocation, and monitoring network performance. The

TelecommunicationCompany class has methods for scaling network capacity, ensuring network security, managing network slices, and monitoring network usage.

The **EndUser** class represents users of the network, who have access to network slices and use network services. The EndUser class has methods for accessing network slices, using network services, reporting issues, and providing feedback.

All classes have a member variable called **networkSlices**, which is an array of NetworkSlice objects. The NetworkSlice class is not shown in the diagram, but it is assumed to exist and contain information about specific network slices.

5.5 Design Details

5.9.1 Novelty

It aims to improve the QoS for dynamic resource allocation using a combination and reinforcement learning. This approach is relatively novel, and there may be opportunities to further innovate and improve the approach.

5.9.2 Innovativeness

The approach involves the use of data augmentation, transfer learning, and ensemble learning techniques to improve the performance of the model. These techniques are innovative and can help to improve the accuracy and robustness of the model

5.9.3 Interoperability

It will need to be compatible with existing network infrastructures and communication protocols. This will require careful consideration of interoperability issues and may require modifications to existing systems.

5.9.4 Performance

It will depend on the quality and quantity of the data, as well as the accuracy of the models. The use of data augmentation, transfer learning, and ensemble learning

techniques can help to improve the performance of the model.

5.9.5 Security

It will need to be secure to prevent unauthorized access to sensitive data and protect against malicious attacks. This will require the use of appropriate encryption and security protocols.

5.9.6 Reliability

It will need to be reliable to ensure that it can provide consistent and accurate results over time. This will require the use of robust models and testing procedures to ensure that the system is reliable.

5.9.7 Maintainability

It will need to be maintainable to ensure that it can be updated and modified over time. This will require careful documentation and version control procedures.

5.9.8 Portability

It will need to be portable to allow it to be used in different environments and systems. This will require careful consideration of platform dependencies and system requirements.

5.9.9 Legacy to modernization

It may require the modernization of existing systems to ensure that they are compatible with the new approach. This may require the use of legacy integration techniques and careful testing procedures.

5.9.10 Reusability

It should be designed to be reusable to allow it to be applied to different use cases and scenarios. This will require the use of modular design principles and careful documentation.

5.9.11 Application compatibility

It should be compatible with existing applications and systems to ensure that it can be integrated into existing workflows. This will require careful consideration of application dependencies and compatibility issues.

5.6 Resource utilization

1. **Resource Management Systems**: The proposed approach involves resource allocation in 5G networks, which requires effective management of network resources such as bandwidth, latency, and processing power. Resource management systems are critical to ensure that resources are allocated efficiently and effectively.
2. **Cloud Computing Platforms**: The use of cloud computing platforms can provide significant benefits for resource allocation in 5G networks. Cloud platforms can provide on-demand access to computing resources, which can help meet the dynamic resource requirements of different applications.
3. **Network Function Virtualization (NFV):** NFV is an approach to network architecture that involves virtualizing network functions, such as routing, firewalling, and load balancing. NFV can help improve resource utilization in 5G networks by allowing network functions to be deployed more efficiently.
4. **Edge Computing:** Edge computing involves processing data closer to the source of the data, rather than sending it to a centralized data center. This can help reduce latency and improve performance, which is particularly important for applications that require real-time data processing, such as healthcare.
5. **Machine Learning Platforms:** The proposed approach involves the use of machine learning algorithms for resource allocation. Machine learning platforms provide a range of tools and services for developing, deploying, and managing machine learning models.
6. **DevOps Processes**: The development and deployment of machine learning models require a range of processes, including version control, continuous integration, and continuous deployment. DevOps processes can help streamline the development and deployment of machine learning models.

CHAPTER 6

PROPOSED METHODOLOGY

6.1 Approach 1: Using Traditional Resource Allocation Algorithms

The traditional 5G network slicing approach employs diverse resource allocation methods, with our discussion centered on two specific algorithms.

A. **Dynamic Allocation**: This real-time resource allocation method dynamically adjusts resources based on evolving application demands, ensuring adaptive and efficient utilization.

B. **Priority-based allocation**: This resource allocation method assigns resources based on application priority levels, prioritizing higher-priority applications for preferential access. The strategy aligns resource allocation with the criticality or importance of each application in the system.

By combining both algorithms, we integrated a hybrid approach that incorporates both dynamic resource allocation, responsive to the real-time demands of applications, and priority-based allocation, ensuring resources are assigned in accordance with application priority levels. This synergy creates a comprehensive resource allocation strategy that adapts to evolving requirements while strategically prioritizing higher-criticality applications for optimal system performance.

In a 5G network simulation, the BaseStation is central, overseeing coverage areas, capacities, and network slices, dynamically allocating resources for client connections. Clients, as user devices, exhibit modeled behaviors and connect/disconnect from slices, consuming resources. The Coverage entity defines a base station's spatial extent. The Distributor generates random values for mobility patterns and usage frequencies. Network Slices, within the Slice entity, manage users, bandwidth, and capacities. Resource containers, represented by the Container entity, model capacity within slices for resource allocation.

The KDTree, implementing a k-d tree, is a crucial tool for spatial searches. It enhances efficiency by swiftly identifying the closest base stations to clients, facilitating optimized connectivity within the simulated network environment.

Together, these interconnected components form a comprehensive and dynamic simulation framework, enabling the exploration of various scenarios and behaviours within the context of a 5G network.

These are the results obtained after performing a hybrid approach:

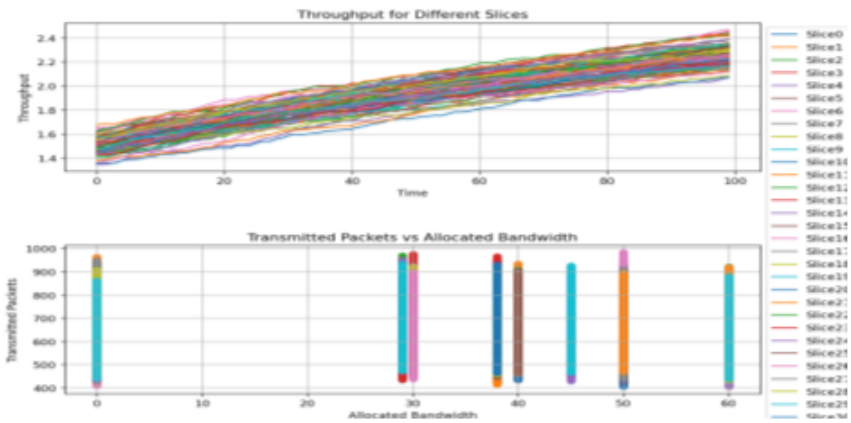


Fig-2 :Priority Based Allocation

Fig. 2 showcases prioritized resource distribution based on priority levels. Higher-priority slices receive increased resources, ensuring critical segments receive the necessary allocation.

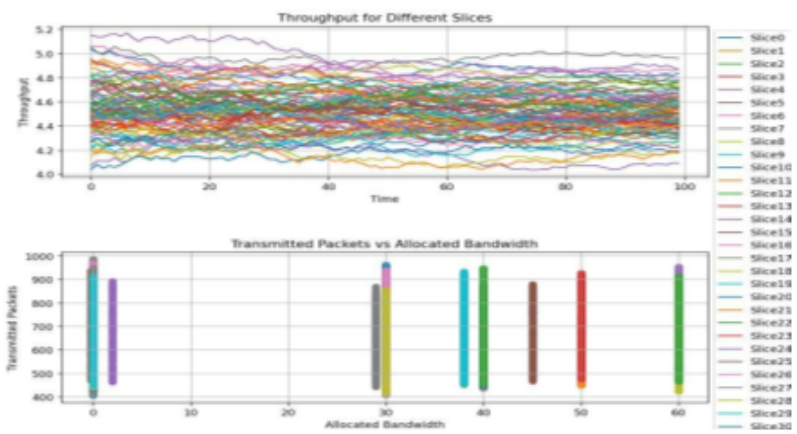


Fig-3: Dynamic Resource Allocation

Fig.2. demonstrates the dynamic adaptation to evolving network conditions. It visualizes real-time adjustments in resource allocation to slices, catering to their

individual requirements. This promotes adaptability and responsiveness in the allocation system

6.1Approach 2: Using Machine Learning Algorithms

1) Random Forest Classifier:

- Ensemble learning method using a collection of decision trees.
- Robust to overfitting and handles missing values well.

2) AdaBoost Classifier:

- Boosting algorithm that combines weak learners into a strong one.
- Adaptive, focuses on misclassified samples.

3) Gradient Boosting Classifier:

- Sequentially builds trees, correcting errors of the previous ones.
- Robust to outliers and handles different types of data well.

4) Extra Trees Classifier:

- Similar to Random Forest but builds trees with random splits.
- Fast training and prediction, suitable for large datasets.

5) Bagging Classifier:

- Reduces variance and overfitting by training multiple models.
- Improves stability and generalization, works well with unstable models.

6) Decision Tree Classifier:

- Simple and interpretable model for classification and regression.
- Prone to overfitting but suitable when interpretability is crucial.

7) MLP Classifier (Multi-Layer Perceptron):

- Deep learning model capable of learning complex relationships.
- Suitable for complex classification and regression tasks.

8) Stacking Classifier:

- Ensemble learning technique that combines predictions from multiple models.

- Uses a meta-model to make the final prediction.

9) Voting Classifier:

- Combines predictions from multiple models by majority voting (hard voting) or weighted voting (soft voting).
- Ensemble method for improved overall performance.

10) LSTM - Through sequential learning of network parameters, this model harnesses historical data patterns of Packet Loss Rate, Packet delay, slice Type, Time, and LTE/5g Category. Its sliding window approach and extensive epoch-based training enable precise predictions of future resource demands, empowering proactive and adaptive resource allocation strategies in dynamic 5G network slicing scenarios.

Project Workflow

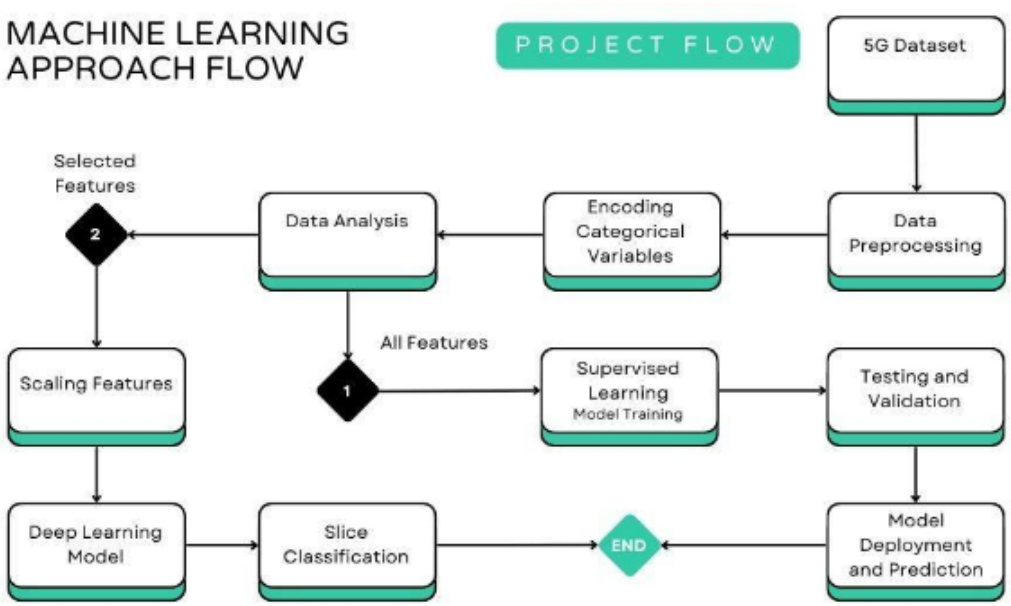


Fig-4: Machine Learning Approach Workflow

CHAPTER 7

IMPLEMENTATION AND PSEUDOCODE

7.1 Algorithm and Pseudocode

7.2 Implementation and Results

SIMULATION

Pseudocode:

```
class DynamicCapacityAllocator:
    def __init__(self, slices, switches):
        self.slices = slices
        self.switches = switches

    def allocate_resources(self):
        for network_slice in self.slices:
            required_bandwidth = network_slice.qos_requirements['bandwidth']

            for switch in self.switches:
                allocated_bandwidth = min(required_bandwidth, switch.capacity)
                switch.capacity -= allocated_bandwidth
                network_slice.allocated_bandwidth = allocated_bandwidth
                print(f"DynamicCapacityAllocator--Allocated {allocated_bandwidth} bandwidth to {network_slice.name}")
                if allocated_bandwidth >= switch.capacity * 0.8:
                    switch.capacity = switch.capacity - (allocated_bandwidth * 0.1)
                break
            else:
                print(f"DynamicCapacityAllocator--Insufficient bandwidth to allocate to {network_slice.name}")

    def deallocate_resources(self):
        for network_slice in self.slices:
            allocated_bandwidth = network_slice.allocated_bandwidth
            for switch in self.switches:
                if switch.capacity + allocated_bandwidth <= switch.initial_capacity:
                    switch.capacity += allocated_bandwidth
                    network_slice.allocated_bandwidth = 0
                    print(f"DynamicCapacityAllocator--Deallocated {allocated_bandwidth} bandwidth from {network_slice.name}")
                    break
            else:
                print(f"DynamicCapacityAllocator--Unable to deallocate bandwidth from {network_slice.name}")
```

Fig-5:Dynamic Capacity Allocation Pseudocode Diagram

Dynamic Capacity Allocator manages the allocation and deallocation of network bandwidth between network slices and switches. The allocate_resources method allocates bandwidth to slices while monitoring switch capacity, potentially reducing it by 10% if bandwidth usage exceeds 80%. The

deallocate_resources method returns bandwidth to switches and logs an error if it can't deallocate without exceeding the initial capacity.

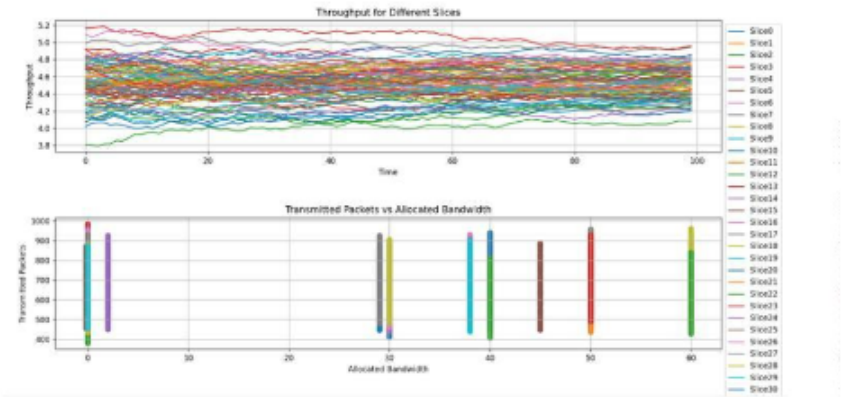


Fig-6: Dynamic Allocation Results

Pseudocode:

```
class WFQResourceAllocator:
    def __init__(self, slices, switches):
        self.slices = slices
        self.switches = switches

    def allocate_resources(self):
        total_capacity = sum(switch.capacity for switch in self.switches)

        for network_slice in self.slices:
            required_bandwidth = network_slice.qos_requirements['bandwidth']
            allocated_bandwidth = int(total_capacity * (required_bandwidth / total_capacity))

            for switch in self.switches:
                if allocated_bandwidth <= switch.capacity:
                    switch.capacity -= allocated_bandwidth
                    network_slice.allocated_bandwidth = allocated_bandwidth
                    print(f"WFQResourceAllocator--Allocated {allocated_bandwidth} bandwidth to {network_slice.name}")
                    break
            else:
                print(f"WFQResourceAllocator--Insufficient bandwidth to allocate to {network_slice.name}")
```

Fig-7: Weighted Fair Queueing Pseudocode diagram

The **class WFQResourceAllocator** allocates bandwidth to network slices based on a Weighted Fair Queueing (WFQ) approach, considering the total capacity of switches.

In the allocate_resources method, it calculates the total capacity by summing the capacities of all switches and then allocates bandwidth to network slices proportionally. It ensures that the allocated bandwidth doesn't exceed the available capacity of switches and logs an error message if insufficient bandwidth is available.

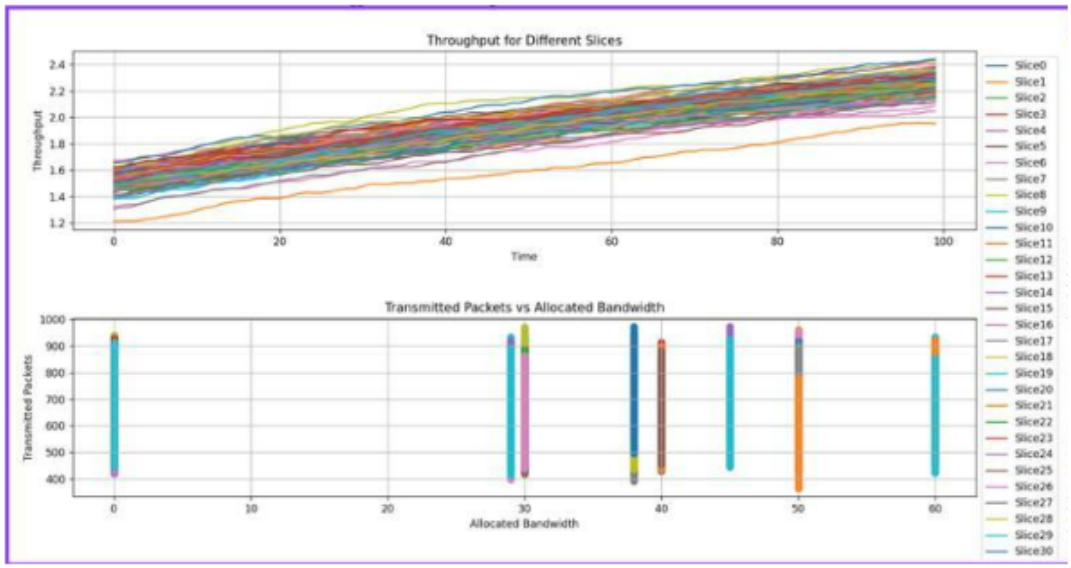


Fig-8: Weighted Fair Queue Allocation Results

Pseudocode:

```
class PriorityBasedAllocator:
    def __init__(self, slices, switches):
        self.slices = slices
        self.switches = switches

    def allocate_resources(self):
        priority_slices = sorted(self.slices, key=lambda slice: slice.qos_requirements['priority'], reverse=True)

        for network_slice in priority_slices:
            required_bandwidth = network_slice.qos_requirements['bandwidth']

            for switch in self.switches:
                if required_bandwidth <= switch.capacity:
                    switch.capacity -= required_bandwidth
                    network_slice.allocated_bandwidth = required_bandwidth
                    print(f"PriorityBasedAllocator--Allocated {required_bandwidth} bandwidth to {network_slice.name}")
                    break
            else:
                print(f"PriorityBasedAllocator--Insufficient bandwidth to allocate to {network_slice.name}")
```

Fig-9: Priority Based Allocation Pseudocode diagram

PriorityBasedAllocator that allocates bandwidth to network slices based on their priority. It sorts the slices by priority in decreasing order.

In the allocate_resources method, it allocates bandwidth to network slices starting from the highest priority slices. It checks if the required bandwidth is available in switches and logs an error message if there's insufficient bandwidth for a slice.

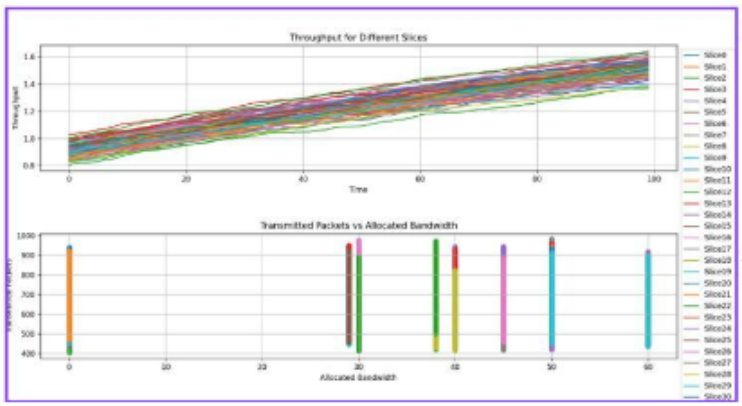


Fig-10: Priority Allocation Results

The three provided resource allocation algorithms cater to different scenarios and priorities:

DynamicCapacityAllocator: Best for dynamic environments with fluctuating bandwidth needs, such as real-time streaming services.

WFQResourceAllocator: Ideal for fair resource sharing in cloud computing environments.

PriorityBasedAllocator: Suitable for applications with strict priority requirements, like emergency services or mission-critical systems.

Choice of Selection for Testing in 5G Architecture with Simulation:

Dynamic Resource Allocation is the preferred choice for the 5G architecture due to its adaptability to changing conditions, efficient resource utilization, and responsiveness to demand fluctuations, and the ability to optimize Quality of Service (QoS). In a dynamic and diverse 5G environment, this approach ensures fair and effective distribution of resources across different slices, making it well-suited for the network's varying and evolving demands. While Weighted Fairness, Priority-Based Allocation, and Proportional Fairness have merits, Dynamic Resource Allocation's flexibility and responsiveness make it better aligned with the dynamic nature of 5G networks. The specific choice depends on the unique characteristics and goals of the 5G network under consideration.

Let's setup the 5g architecture for Simulation with python:

1. BaseStation:

Purpose: Represents the base station in a 5G network.

Functionality:

Manages coverage area, capacity, and slices.

Handles dynamic allocation and deallocation of resources.

Participates in client connection and disconnection.

Pseudocode:

```
class BaseStation:
    def __init__(self, pk, coverage, capacity_bandwidth, slices=None):
        self.pk = pk
        self.coverage = coverage
        self.capacity_bandwidth = capacity_bandwidth
        self.slices = slices
        print(self)

    def __str__(self):
        return f'BS_{self.pk:<2}\t cov:{self.coverage}\t with cap {self.capacity_bandwidth:<5}'
```

Fig-11: Setting up Base Station pseudocode diagram

2. Client:
Purpose: Represents a client device in the 5G network.
Functionality:
Models client behaviour, mobility, and usage patterns.
Connects and disconnects from network slices.
Consumes resources and tracks usage statistics.

Pseudocode:

```
import operator
import random

from .utils import distance, KDTree


class client:
    def __init__(self, pk, env, x, y, mobility_pattern, usage_freq,
                 subscribed_slice_index, stat_collector,
                 base_station=None):
        self.pk = pk
        self.env = env
        self.x = x
        self.y = y
        self.mobility_pattern = mobility_pattern
        self.usage_freq = usage_freq
        self.base_station = base_station
        self.stat_collector = stat_collector
        self.subscribed_slice_index = subscribed_slice_index
        self.usage_remaining = 0
        self.last_usage = 0
        self.closest_base_stations = []
        self.connected = False

        # stats
        self.total_connected_time = 0
        self.total_unconnected_time = 0
        self.total_request_count = 0
        self.total_consume_time = 0
        self.total_usage = 0

        self.action = env.process(self.iter())
        # print(self.usage_freq)
def iter(self):
    '''
    There are four steps in a cycle:
        1- .00: Lock
        2- .25: Stats
        3- .50: Release
        4- .75: Move
    '''

    # .00: Lock
    if self.base_station is not None:
        if self.usage_remaining > 0:
            if self.connected:
                self.start_consume()
            else:
                self.connect()
        else:
            if self.connected:
                self.disconnect()
            else:
                self.generate_usage_and_connect()

    yield self.env.timeout(0.25)

    # .25: Stats

    yield self.env.timeout(0.25)

    # .50: Release
    # Base station check skipped as it's already implied by self.connected
    if self.connected and self.last_usage > 0:
        self.release_consume()
        if self.usage_remaining <= 0:
            self.disconnect()

    yield self.env.timeout(0.25)
```

Fig-12: Client Device pseudocode diagram

```python
def get_slice(self):
    if self.base_station is None:
        return None
    return self.base_station.slices[self.subscribed_slice_index]

def generate_usage_and_connect(self):
    if self.usage_freq < random.random() and self.get_slice() is not None:
        # Generate a new usage
        self.usage_remaining = self.get_slice().usage_pattern.generate()
        self.total_request_count += 1
        self.connect()
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] requests {self.usage_remaining} usage.')

def connect(self):
    s = self.get_slice()
    if self.connected:
        return
    # increment connect attempt
    self.stat_collector.incr_connect_attempt(self)
    if s.is_avaliable():
        s.connected_users += 1
        self.connected = True
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] connected to slice={self.get_slice()} @ {self.base_station}')
        return True
    else:
        self.assign_closest_base_station(exclude=[self.base_station.pk])
        if self.base_station is not None and self.get_slice().is_avaliable():
            # handover
            self.stat_collector.incr_handover_count(self)
        elif self.base_station is not None:
            # block
            self.stat_collector.incr_block_count(self)
        else:
            pass # uncovered
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] connection refused to slice={self.get_slice()} @ {self.base_stati
        return False
def disconnect(self):
    if self.connected == False:
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] is already disconnected from slice={self.get_slice()} @ {self.base_st
    else:
        slice = self.get_slice()
        slice.connected_users -= 1
        self.connected = False
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] disconnected from slice={self.get_slice()} @ {self.base_station}')
    return not self.connected

def start_consume(self):
    s = self.get_slice()
    amount = min(s.get_consumable_share(), self.usage_remaining)
    # Allocate resource and consume ongoing usage with given bandwidth
    s.capacity.get(amount)
    print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] gets {amount} usage.')
    self.last_usage = amount

def release_consume(self):
    s = self.get_slice()
    # Put the resource back
    if self.last_usage > 0: # note: s.capacity.put cannot take 0
        s.capacity.put(self.last_usage)
        print(f'[{int(self.env.now)}] Client_{self.pk} [{self.x}, {self.y}] puts back {self.last_usage} usage.')
        self.total_consume_time += 1
        self.total_usage += self.last_usage
        self.usage_remaining -= self.last_usage
        self.last_usage = 0
```

```python
# Check closest base_stations of a client and assign the closest non-excluded avaliable base_station to the client.
def assign_closest_base_station(self, exclude=None):
    updated_list = []
    for d,b in self.closest_base_stations:
        if exclude is not None and b.pk in exclude:
            continue
        d = distance((self.x, self.y), (b.coverage.center[0], b.coverage.center[1]))
        updated_list.append((d,b))
    updated_list.sort(key=operator.itemgetter(0))
    for d,b in updated_list:
        if d <= b.coverage.radius:
            self.base_station = b
            print(f'[{int(self.env.now)}] Client_{self.pk} freshly assigned to {self.base_station}')
            return
    if KDTree.last_run_time is not int(self.env.now):
        KDTree.run(self.stat_collector.clients, self.stat_collector.base_stations, int(self.env.now), assign=False)
    self.base_station = None

def __str__(self):
    return f'Client_{self.pk} [{self.x:<5}, {self.y:>5}] connected to: slice={self.get_slice()} @ {self.base_station}\t with mobility pattern of
```

3. Coverage:
Purpose: Represents the coverage area of a base station.
Functionality:
Defines the center and radius of the circular coverage area.
Checks if a given point is within the coverage area.

Pseudocode:

```
import math
class Coverage:
    def __init__(self, center, radius):
        self.center = center
        self.radius = radius

    def _get_gaussian_distance(self, p):
        return math.sqrt(sum((i-j)**2 for i,j in zip(p, self.center)))

    def is_in_coverage(self, x, y):
        return self._get_gaussian_distance((x,y)) <= self.radius

    def __str__(self):
        x, y = self.center
        return f'[c=({x:<4}, {y:>4}), r={self.radius:>4}]'
```

Fig-13: Coverage area pseudocode diagram

4. Distributor:
Purpose: Generates random values based on specified distributions.
Functionality:
Used for generating random mobility patterns, usage frequencies, etc.

Pseudocode:

```
class Distributor:
    def __init__(self, name, distribution, *dist_params, divide_scale=1):
        self.name = name
        self.distribution = distribution
        self.dist_params = dist_params
        self.divide_scale = divide_scale

    def generate(self):
        return self.distribution(*self.dist_params)

    def generate_scaled(self):
        return self.distribution(*self.dist_params) / self.divide_scale

    def generate_movement(self):
        x = self.distribution(*self.dist_params) / self.divide_scale
        y = self.distribution(*self.dist_params) / self.divide_scale
        return x, y

    def __str__(self):
        return f'[{self.name}: {self.distribution.__name__}: {self.dist_params}]'
```

Fig-14: Distributer pseudocode

5. Slice:
Purpose: Represents a network slice with specified parameters.
Functionality:
Manages connected users, bandwidth guarantees, and capacities.

Provides methods for calculating consumable shares and availability.
Pseudocode:

```
class Slice:
    def __init__(self, name, ratio,
                 connected_users, user_share, delay_tolerance, qos_class,
                 bandwidth_guaranteed, bandwidth_max, init_capacity,
                 usage_pattern):
        self.name = name
        self.connected_users = connected_users
        self.user_share = user_share
        self.delay_tolerance = delay_tolerance
        self.qos_class = qos_class
        self.ratio = ratio
        self.bandwidth_guaranteed = bandwidth_guaranteed
        self.bandwidth_max = bandwidth_max
        self.init_capacity = init_capacity
        self.capacity = 0
        self.usage_pattern = usage_pattern

    def get_consumable_share(self):
        if self.connected_users <= 0:
            return min(self.init_capacity, self.bandwidth_max)
        else:
            return min(self.init_capacity/self.connected_users, self.bandwidth_max)

    def is_avaliable(self):
        real_cap = min(self.init_capacity, self.bandwidth_max)
        bandwidth_next = real_cap / (self.connected_users + 1)
        if bandwidth_next < self.bandwidth_guaranteed:
            return False
        return True

    def __str__(self):
        return f'{self.name:<10} init={self.init_capacity:<5} cap={self.capacity.level:<5}
        diff={(self.init_capacity - self.capacity.level):<5}'
```

Fig-15: Network Slice Parameters

6. Container:

Purpose: Represents a resource container with a specified capacity.

Functionality:

Used for modeling resource capacity within slices.

Pseudocode:

```
class Container:
    def __init__(self, init, capacity):
        self.capacity = capacity
        self.level = init

    def get(self, amount):
        if amount <= self.level:
            self.level -= amount
            return True
        else:
            return False

    def put(self, amount):
        if amount + self.level <= self.capacity:
            self.level += amount
            return True
        else:
            return False
```

Fig-16: Container Pseudocode

1. KD Tree:
Purpose: Implements a k-d tree for efficient spatial searches.

Functionality:

Used for finding the closest base stations to clients.

Pseudocode:

```python
# Initial connections using k-d tree
def kdtree(clients, base_stations):

    c_coor = [(c.x,c.y) for c in clients]
    bs_coor = [p.coverage.center for p in base_stations]

    tree = KDTree(bs_coor, leaf_size=2)
    res = tree.query(c_coor)

    for c, d, p in zip(clients, res[0], res[1]):
        if d[0] <= base_stations[p[0]].coverage.radius:
            c.base_station = base_stations[p[0]]

class KDTree:
    last_run_time = 0
    limit = None

    # Initial connections using k-d tree
    @staticmethod
    def run(clients, base_stations, run_at, assign=True):
        print(f'KDTREE CALL [{run_at}] - limit: {KDTree.limit}')
        if run_at == KDTree.last_run_time:
            return
        KDTree.last_run_time = run_at

        c_coor = [(c.x,c.y) for c in clients]
        bs_coor = [p.coverage.center for p in base_stations]

        tree = kdt(bs_coor, leaf_size=2)
        res = tree.query(c_coor,k=min(KDTree.limit,len(base_stations)))

        # print(res[0])
        for c, d, p in zip(clients, res[0], res[1]):
            if assign and d[0] <= base_stations[p[0]].coverage.radius:
                c.base_station = base_stations[p[0]]
            c.closest_base_stations = [(a, base_stations[b]) for a,b in zip(d,p)]
```

Fig-17: k-d Tree for efficient spatial searches

Simulation input and conditions:

1.Network Slices:Four network slices are considered: x_eMBB, x_mMTC, x_URLLC, x_voice, y_eMBB, y_eMBB_p, and y_voice.

Each slice has specific delay tolerances, QoS classes, bandwidth guarantees, and usage patterns.

2. Base Stations: Multiple base stations with varying coverage areas, capacities, and ratios for each network slice. Base stations are strategically located with different x and y coordinates.

3. Mobility Patterns: Clients exhibit different mobility patterns, including car, walk, stationary, tram, and slackperson. Mobility patterns have associated distributions and client weights.

4. Client Characteristics: Clients have random locations within specified x and y ranges. Usage frequencies for clients are generated with a specified distribution.

5. Simulation Settings:

The simulation runs for a specified time (100 units).

The number of clients is set to 100.

SAMPLE OUTPUT WITH DYNAMICALLY ALLOCATED RESOURCES :

```
BS_0     cov:[c=(182 , 1414), r= 224]    with cap 20000000000
BS_1     cov:[c=(44  , 1916), r= 368]    with cap 50
BS_2     cov:[c=(544 , 1714), r= 334]    with cap 25000000000
BS_3     cov:[c=(556 , 1262), r= 250]    with cap 20000000000
BS_4     cov:[c=(126 , 1016), r= 384]    with cap 30000000000
KDTREE CALL [0] - limit: 5
KDTREE CALL [1] - limit: 5
[1] Client_2 freshly assigned to BS_2    cov:[c=(544 , 1714), r= 334]    with cap 25000000000
[1] Client_11 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_13 freshly assigned to BS_2   cov:[c=(544 , 1714), r= 334]    with cap 25000000000
[1] Client_15 freshly assigned to BS_1   cov:[c=(44  , 1916), r= 368]    with cap 50
[1] Client_16 freshly assigned to BS_2   cov:[c=(544 , 1714), r= 334]    with cap 25000000000
[1] Client_21 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_22 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_25 freshly assigned to BS_1   cov:[c=(44  , 1916), r= 368]    with cap 50
[1] Client_26 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_31 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_35 freshly assigned to BS_1   cov:[c=(44  , 1916), r= 368]    with cap 50
[1] Client_38 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_39 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_40 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_43 freshly assigned to BS_2   cov:[c=(544 , 1714), r= 334]    with cap 25000000000
[1] Client_44 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_45 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_49 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_50 freshly assigned to BS_1   cov:[c=(44  , 1916), r= 368]    with cap 50
[1] Client_60 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_66 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_70 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_77 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_79 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_83 freshly assigned to BS_2   cov:[c=(544 , 1714), r= 334]    with cap 25000000000
[1] Client_86 freshly assigned to BS_3   cov:[c=(556 , 1262), r= 250]    with cap 20000000000
[1] Client_88 freshly assigned to BS_0   cov:[c=(182 , 1414), r= 224]    with cap 20000000000
[1] Client_95 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
[1] Client_97 freshly assigned to BS_4   cov:[c=(126 , 1016), r= 384]    with cap 30000000000
```

Fig-18: Dynamically Allocated Resources

```
[2] Client_2 [363.0, 1736.0] connected to slice=x_eMBB     init=10000000000.0 cap=10000000000.0 diff=0.0   @ BS_2    cov:[c=(544 , 1714), r= 334
[2] Client_2 [363.0, 1736.0] requests 675780921 usage.
[2] Client_11 [593.2250345150521, 1244.1351186329468] connected to slice=x_eMBB     init=7400000000.0 cap=7400000000.0 diff=0.0   @ BS_3     cov
[2] Client_11 [593.2250345150521, 1244.1351186329468] requests 409345437 usage.
[2] Client_13 [726.1933179062951, 1920.51162572711] connected to slice=y_voice    init=750000000.0 cap=750000000.0 diff=0.0   @ BS_2     cov:[c=
[2] Client_13 [726.1933179062951, 1920.51162572711] requests 6114555 usage.
KDTREE CALL [2] - limit: 5
[2] Client_15 [171.0, 1712.0] connection refused to slice=None @ None
[2] Client_15 [171.0, 1712.0] requests 4403338 usage.
[2] Client_16 [655.0321180444766, 1941.0146079692865] connected to slice=y_eMBB_p   init=2500000000.0 cap=2500000000.0 diff=0.0   @ BS_2     cov
[2] Client_16 [655.0321180444766, 1941.0146079692865] requests 587119486 usage.
[2] Client_21 [108.0, 1568.0] connected to slice=x_eMBB     init=11800000000.0 cap=11800000000.0 diff=0.0   @ BS_0   cov:[c=(182 , 1414), r= 224
[2] Client_21 [108.0, 1568.0] requests 230652138 usage.
[2] Client_22 [429.9692110997957, 1307.9479548188738] connected to slice=y_eMBB     init=8000000000.0 cap=8000000000.0 diff=0.0   @ BS_3     cov
[2] Client_22 [429.9692110997957, 1307.9479548188738] requests 163703164 usage.
[2] Client_25 [113.0, 1947.0] connection refused to slice=None @ None
[2] Client_25 [113.0, 1947.0] requests 2475963 usage.
[2] Client_26 [751.0, 1386.0] connected to slice=y_eMBB     init=8000000000.0 cap=8000000000.0 diff=0.0   @ BS_3     cov:[c=(556 , 1262), r= 250
[2] Client_26 [751.0, 1386.0] requests 234093697 usage.
[2] Client_31 [86.0, 1201.0] connected to slice=x_eMBB     init=6000000000.0 cap=6000000000.0 diff=0.0   @ BS_4     cov:[c=(126 , 1016), r= 384
[2] Client_31 [86.0, 1201.0] requests 67176664 usage.
[2] Client_35 [108.0, 1782.0] connected to slice=x_eMBB     init=20.0  cap=20.0  diff=0.0   @ BS_1   cov:[c=(44  , 1916), r= 368]    with cap 50
[2] Client_35 [108.0, 1782.0] requests 81538112 usage.
[2] Client_38 [264.05087606795996, 1305.0367795240354] connected to slice=y_voice    init=1000000000.0 cap=1000000000.0 diff=0.0   @ BS_0    cov
[2] Client_38 [264.05087606795996, 1305.0367795240354] requests 6101494 usage.
[2] Client_39 [364.0, 728.0] connected to slice=y_eMBB     init=7500000000.0 cap=7500000000.0 diff=0.0   @ BS_4     cov:[c=(126 , 1016), r= 384
[2] Client_39 [364.0, 728.0] requests 433506826 usage.
[2] Client_40 [265.0, 1594.0] connected to slice=y_eMBB_p   init=1000000000.0 cap=1000000000.0 diff=0.0   @ BS_0     cov:[c=(182 , 1414), r= 224
[2] Client_40 [265.0, 1594.0] requests 652446379 usage.
[2] Client_43 [353.0, 1748.0] connected to slice=y_eMBB     init=9500000000.0 cap=9500000000.0 diff=0.0   @ BS_2     cov:[c=(544 , 1714), r= 334
[2] Client_43 [353.0, 1748.0] requests 108153252 usage.
[2] Client_44 [479.8845324400094, 1416.9897730301977] connected to slice=y_eMBB     init=8000000000.0 cap=8000000000.0 diff=0.0   @ BS_3     cov
[2] Client_44 [479.8845324400094, 1416.9897730301977] requests 398621980 usage.
[2] Client_45 [459.74917018801045, 887.1918682555453] connected to slice=x_URLLC    init=1500000000.0 cap=1500000000.0 diff=0.0   @ BS_4     cov
[2] Client_45 [459.74917018801045, 887.1918682555453] requests 3009555 usage.
[2] Client_49 [270.6243117518809, 1061.1896539128002] connected to slice=x_voice    init=4500000000.0 cap=4500000000.0 diff=0.0   @ BS_4     cov
```

Fig-19: Dynamic Allocation Results

Inputs were tested with 4 different networks each with different scenarios:

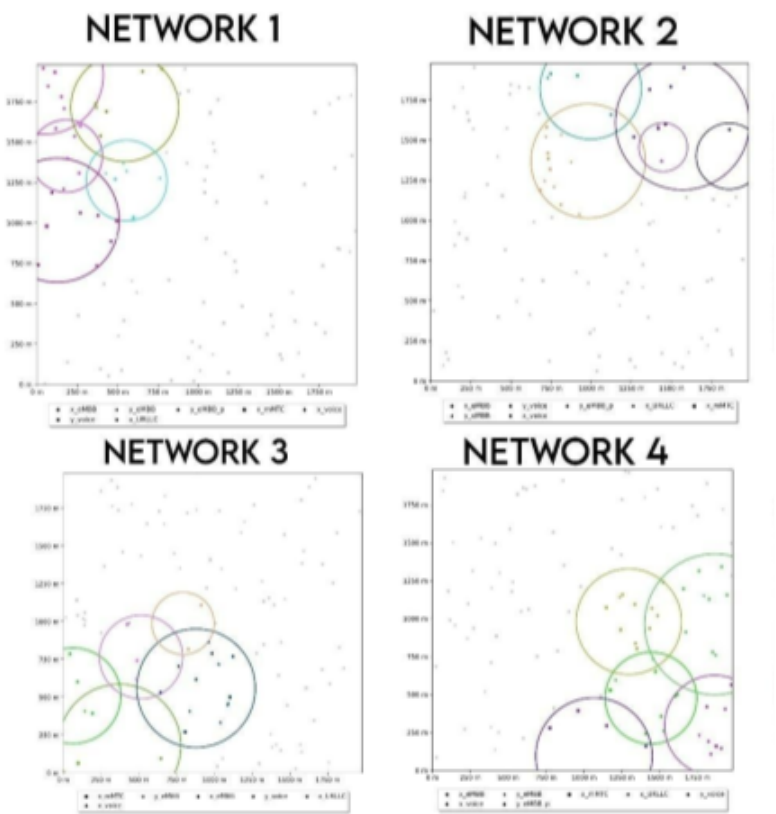


Fig-20: Four Different Networ Scenarios

Heterogenous network(hetnet)

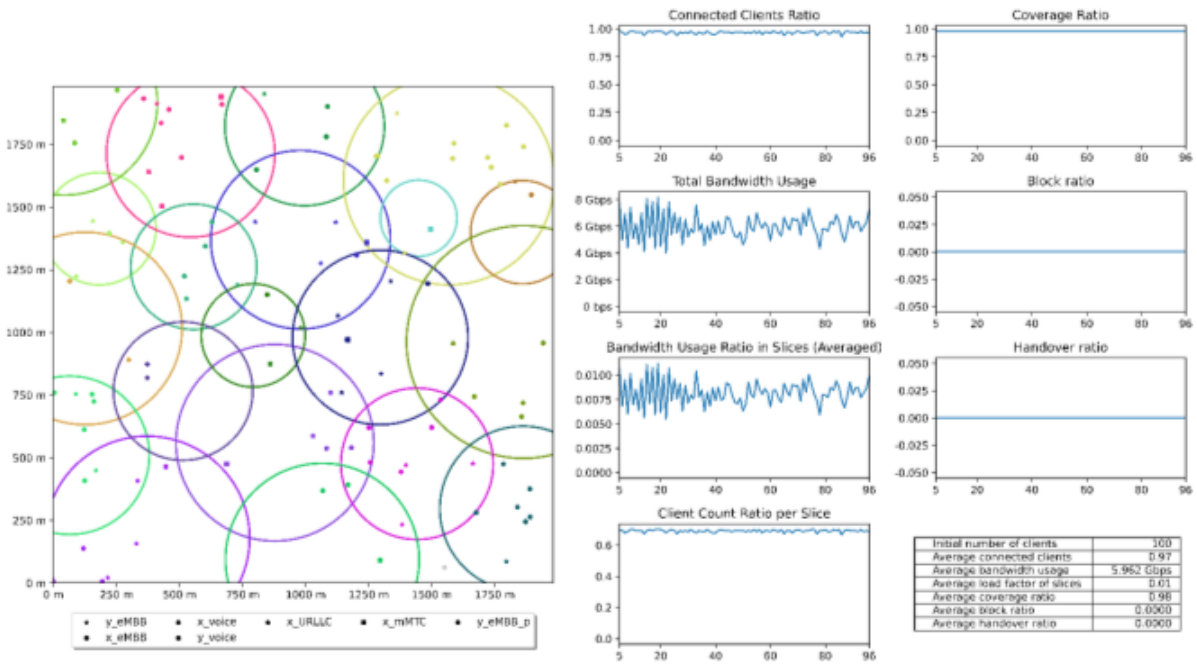


Fig-21: Heterogeneous network

Networks	N1	N2	N3	N4
Coverage Ratio	0.28	0.24	0.26	0.44
Avg. Connected Clients	28	24	25	43
Avg bandwidth (Gbps)	1.691	1.45	1.033	2.95
Avg Load factor	0.02	0.01	0.01	0.01

Table 1: Simulated results

The table 1 above explains the results in the simulated environment when the best traditional algorithm (which is the Dynamic Allocation Algorithm) is implemented for allocation of resources to the network slices.

The four networks (Network 1, Network 2, Network 3, and Network 4) exhibit variations in their base station structures and configurations. Network 1 and Network 2 feature fixed base stations with specific

characteristics, while Network 3 and Network 4 introduce heterogeneity with multiple base stations, exploring variations in capacity, coverage, and ratios. HetNet, on the other hand, specifically focuses on a heterogeneous network with diverse base station types.

The mobility patterns, including car, walk, stationary, tram, and slack person, are consistent across all networks, contributing to the randomness of client locations and usage frequencies. Each network serves as a baseline for comparison, with Networks 3 and 4 providing more complexity for a nuanced analysis. The differing base station setups and mobility scenarios are expected to yield insights into potential trade-offs and varying impacts on different use cases and slices, offering a comprehensive exploration of network behaviour in heterogeneous and homogeneous environments

INTRODUCTION TO OUR DATASET

The dataset encompasses vital attributes that define parameters crucial to 5G network slicing. Among these attributes is User Device Type, indicative of diverse connecting devices such as smartphones, tablets, or industrial machinery. Duration records the connection time in seconds, while Packet Loss Rate serves as a metric for connection reliability, where lower rates denote enhanced reliability. Packet Delay Budget signifies the maximum allowable latency, a critical factor for achieving prompt connections. Bandwidth measures the transferable data per unit of time, and Delay Rate and Speed delineate connection speed and packet delay rates, respectively. Jitter quantifies the variation in packet delay, while Modulation Type specifies the encoding method for transmission. Finally, Slice Type categorizes slices based on performance needs, encompassing factors such as latency, throughput, and security.

Exploratory data analysis has been conducted with the aim of discerning the attribute that exerts the most significant influence in optimizing resource allocation. This optimization is crucial for meeting diverse service-level agreements across distinct network slices, thereby enhancing the efficiency and effectiveness of 5G network operations.

User Device Type:	This attribute indicates the type of device that is used to connect to the network slice.
Duration (sec):	The duration of the network slice connection, in seconds.
Packet Loss Rate (Reliability):	The percentage of packets that are lost during the network slice connection.
Packet Delay Budget (Latency (ms):	The maximum amount of latency that is allowed for the network slice connection.
Bandwidth (GHz):	Bandwidth is the amount of data that can be transferred per unit Time.
Delay Rate (Mbps):	Delay rate is the rate at which packets are delayed during the network slice connection.
Speed (Mbps):	Speed is the maximum amount of data that can be transferred per unit of time.
Jitter (ps):	Jitter is the variation in the delay of packets during the network slice connection
Modulation Type:	Modulation is the process of encoding data into a signal that can be transmitted over a network

Fig-22: 5G Network Slicing Dataset

Machine Learning Model Review:

Machine Learning Algorithms

1) Random Forest Classifier

[60]:

```
# random forest classifier
param_grid = {
    'bootstrap': [False,True],
    'max_depth': [5,8,10, 20],
    'max_features': [3, 4, 5, None],
    'min_samples_split': [2, 10, 12],
    'n_estimators': [100, 200, 300]
}

rfc = RandomForestClassifier()

grid_random_forest = GridSearchCV(estimator = rfc, param_grid = param_grid, cv = 5,

grid_random_forest.fit(X_train,y_train)
y_pred_rfc = grid_random_forest.predict(X_test)
best_random_forest_estimator = grid_random_forest.best_estimator_
print("Accuracy: ",accuracy_score(y_test,y_pred_rfc))
print(grid_random_forest.best_params_)
print(grid_random_forest.best_estimator_)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Accuracy:  0.7266666666666667
{'bootstrap': True, 'max_depth': 8, 'max_features': None, 'min_samples_split': 2, 'n_estimators': 20
0}
RandomForestClassifier(max_depth=8, max_features=None, n_estimators=200)
```

Fig-23: Random Forest Classifier

2) AdaBoost Classifier

```
# AdaBoost Classifier
param_grid_adaboost = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1]
}

# Replace 'base_estimator' with 'estimator'
adaboost = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=5), random_
grid_adaboost = GridSearchCV(estimator=adaboost, param_grid=param_grid_adaboost, cv=
grid_adaboost.fit(X_train, y_train)
best_adaboost_estimator = grid_adaboost.best_estimator_
y_pred_ad = grid_adaboost.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_ad))
print(grid_adaboost.best_params_)
print(grid_adaboost.best_estimator_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Accuracy:  0.725
{'learning_rate': 0.01, 'n_estimators': 50}
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=5),
                   learning_rate=0.01, random_state=42)
```

Fig-24: Adaboost Classifier

3) Gradient Boosting Classifier

[62]:

```
# Gradient Boosting Classifier
param_grid_gradient_boosting = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1],
    'max_depth': [3, 5, 7]
}

gradient_boosting = GradientBoostingClassifier(random_state=42)
grid_gradient_boosting = GridSearchCV(estimator=gradient_boosting, param_grid=param_
grid_gradient_boosting.fit(X_train, y_train)
best_gradient_boosting_estimator = grid_gradient_boosting.best_estimator_
y_pred_gb = grid_gradient_boosting.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_gb))
print(grid_gradient_boosting.best_params_)
print(grid_gradient_boosting.best_estimator_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Accuracy:  0.7266666666666667
{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200}
GradientBoostingClassifier(learning_rate=0.01, n_estimators=200,
                           random_state=42)
```

Fig-25: Gradient Boosting Classifier

4) Extra Trees Classifier

[63]:

```
# Extra Trees Classifier
param_grid_extra_trees = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'max_features': [ 'sqrt', 'log2']
}

extra_trees = ExtraTreesClassifier(random_state=42)
grid_extra_trees = GridSearchCV(estimator=extra_trees, param_grid=param_grid_extra_t
grid_extra_trees.fit(X_train, y_train)
best_extra_trees_estimator = grid_extra_trees.best_estimator_
y_pred_et = grid_extra_trees.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_et))
print(grid_extra_trees.best_params_)
print(grid_extra_trees.best_estimator_)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Accuracy:  0.7166666666666667
{'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 200}
ExtraTreesClassifier(max_depth=10, n_estimators=200, random_state=42)
```

Fig-26: Gradient Boosting Classifier

5) Bagging Classifier

[64]:

```
# Bagging Classifier
param_grid_bagging = {
    'n_estimators': [50, 100, 200],
    'max_samples': [0.5, 1.0],
    'max_features': [0.5, 1.0],
    'bootstrap': [True, False]
}

bagging_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(random_state
grid_bagging = GridSearchCV(estimator=bagging_classifier, param_grid=param_grid_bagg
grid_bagging.fit(X_train, y_train)
best_bagging_estimator = grid_bagging.best_estimator_
y_pred_bg = grid_bagging.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_bg))
print(grid_bagging.best_params_)
print(grid_bagging.best_estimator_)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Accuracy:  0.7
{'bootstrap': True, 'max_features': 1.0, 'max_samples': 0.5, 'n_estimators': 200}
BaggingClassifier(estimator=DecisionTreeClassifier(random_state=42),
                  max_samples=0.5, n_estimators=200, random_state=42)
```

Fig-27: Gradient Boosting Classifier

6) Decision Tree Classifier

[65]:
```
# Decision Tree Classifier
param_grid_decision_tree = {
    'max_depth': [None, 5, 10],
    'max_features': [ 'sqrt', 'log2']
}

decision_tree = DecisionTreeClassifier(random_state=42)
grid_decision_tree = GridSearchCV(estimator=decision_tree, param_grid=param_grid_dec
grid_decision_tree.fit(X_train, y_train)
best_decision_tree_estimator = grid_decision_tree.best_estimator_
y_pred_dt = grid_decision_tree.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_dt))
print(grid_decision_tree.best_params_)
print(grid_decision_tree.best_estimator_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Accuracy:  0.6566666666666666
{'max_depth': 10, 'max_features': 'sqrt'}
DecisionTreeClassifier(max_depth=10, max_features='sqrt', random_state=42)
```

Fig-28: Decision Tree Classifier

7) Multi-Layer Perceptron Classifier

[66]:
```
# MLP Classifier
param_grid_mlp = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam'],
    'max_iter': [100, 200, 300]
}

mlp_classifier = MLPClassifier(random_state=42)
grid_mlp = GridSearchCV(estimator=mlp_classifier, param_grid=param_grid_mlp, cv=5,
grid_mlp.fit(X_train, y_train)
best_mlp_estimator = grid_mlp.best_estimator_
y_pred_mlp = grid_mlp.predict(X_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_mlp))
print(grid_mlp.best_params_)
print(grid_mlp.best_estimator_)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Accuracy:  0.415
{'activation': 'tanh', 'hidden_layer_sizes': (50,), 'max_iter': 100, 'solver': 'adam'}
MLPClassifier(activation='tanh', hidden_layer_sizes=(50,), max_iter=100,
              random_state=42)
```

Fig-29: Multi-Layer Perceptron

Model Performance Results

[69]:
```
print("Random Forest Classifier Accuracy: ",accuracy_random_forest)
print("AdaBoost Classifier Accuracy:", accuracy_adaboost)
print("Gradient Boosting Classifier Accuracy:", accuracy_gradient_boosting)
print("Extra Trees Classifier Accuracy:", accuracy_extra_trees)
print("Bagging Classifier Accuracy:", accuracy_bagging)
print("Decision Tree Classifier Accuracy:", accuracy_decision_tree)
print("MLP Classifier Accuracy:", accuracy_mlp)
```

```
Random Forest Classifier Accuracy:  0.7266666666666667
AdaBoost Classifier Accuracy: 0.725
Gradient Boosting Classifier Accuracy: 0.7266666666666667
Extra Trees Classifier Accuracy: 0.7166666666666667
Bagging Classifier Accuracy: 0.7
Decision Tree Classifier Accuracy: 0.6566666666666666
MLP Classifier Accuracy: 0.415
```

Fig-30: ML Models Performance results

8) Stacking Classifier

```
# Stacking Classifier
stacking_models = [
    ('rfc', random_forest),
    ('adaboost', adaboost),
    ('gradient_boosting', gradient_boosting),
    ('extra_trees', extra_trees),
    ('bagging', bagging_classifier),
    ('decision_tree', decision_tree),
    #('lstm', lstm_model),
    ('mlp', mlp_classifier)
]

stacking_classifier = StackingClassifier(estimators=stacking_models, final_estimator=LogisticRegression(), c
stacking_classifier.fit(X_train_scale, y_train)
```

70]:

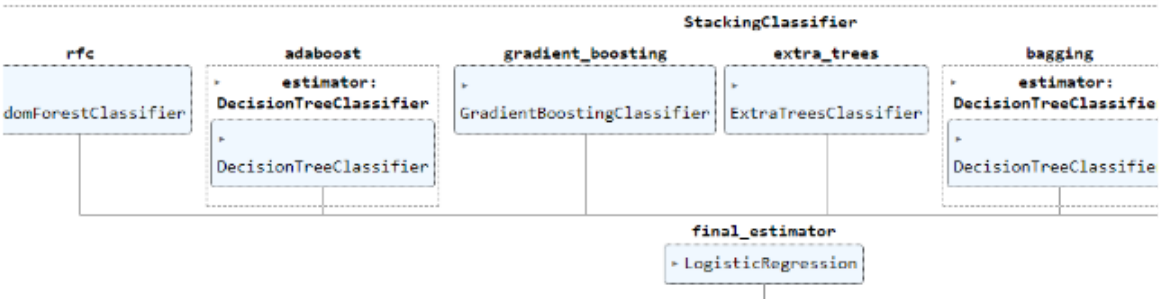


Fig-31: Stacking Classifier

9) Voting Classifier

```
# Hard Voting Classifier
voting_hard = VotingClassifier(estimators=voting_models, voting='hard')
voting_hard.fit(X_train_scale, y_train)
voting_hard_predictions = voting_hard.predict(X_test_scale)
voting_hard_accuracy = accuracy_score(y_test, voting_hard_predictions)
print("Hard Voting Classifier Accuracy:", voting_hard_accuracy)
```

```
Hard Voting Classifier Accuracy: 0.7116666666666667
```

```
# Soft Voting Classifier
voting_soft = VotingClassifier(estimators=voting_models, voting='soft')
voting_soft.fit(X_train_scale, y_train)
voting_soft_predictions = voting_soft.predict(X_test_scale)
voting_soft_accuracy = accuracy_score(y_test, voting_soft_predictions)
print("Soft Voting Classifier Accuracy:", voting_soft_accuracy)
```

```
Soft Voting Classifier Accuracy: 0.6933333333333334
```

Fig:32 Voting Classifier

10) LSTM

```
batch_size = 32
model = Sequential()
# add 1st layer with 68 nodes/perceptons, each will process (2,7) matrix block
model.add(LSTM(68,
           input_shape=(window_length, number_of_features),
           return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(32,
           return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(number_of_features))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

+ Code + Markdown

[81]:

```
model.fit(train, label, batch_size=32, epochs=1000, verbose=1)
```

```
Epoch 1/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.3416 - accuracy: 0.6111
Epoch 2/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.3426 - accuracy: 0.6047
Epoch 3/1000
94/94 [==============================] - 1s 9ms/step - loss: 0.3424 - accuracy: 0.6101
Epoch 4/1000
94/94 [==============================] - 1s 8ms/step - loss: 0.3403 - accuracy: 0.6207
Epoch 5/1000
```

Fig-33: LSTM pseudocode

CHAPTER 8

RESULTS AND DISCUSSION

The performance of various machine learning models was evaluated on the dataset, and the testing accuracies were recorded.

Here are the results:

Algorithm	Accuracy
Random Forest Classifier	72.67%
AdaBoost Classifier	72.50%
Gradient Boosting Classifier	72.67%
Extra Trees Classifier	71.67%
Bagging Classifier	70.00%
Decision Tree Classifier	65.67%
MLP Classifier	41.50%
Stacking Classifier	73.00%
Hard Voting Classifier	71.17%
Soft Voting Classifier	69.33%

Table 2 : Machine Learning Models Results

The Stacking Classifier outperformed individual models, achieving the highest testing accuracy of 73.00%. It indicates that combining the strengths of different models in a stacking ensemble can lead to improved predictive performance.

On the other hand, the MLP Classifier exhibited lower accuracy compared to other models. This could be attributed to factors such as the need for further hyperparameter tuning or potential sensitivity to the specific characteristics of the dataset.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this study, we explored the application of various machine learning models and ensemble techniques for a comprehensive analysis of the dataset. The key findings and conclusions are summarized as follows:

1. **Ensemble Techniques**: Stacking and Voting Classifier approaches were employed to harness the strengths of multiple models. The Stacking Classifier demonstrated superior performance, indicating the potential benefits of combining diverse models.
2. **Model Performance:** Random Forest, AdaBoost, and Gradient Boosting classifiers exhibited competitive testing accuracies, highlighting their effectiveness for the given task. On the contrary, the MLP Classifier showed comparatively lower accuracy, suggesting the need for further optimization.
3. **Stacking Classifier:** The Stacking Classifier emerged as the top-performing model, showcasing the effectiveness of combining multiple models to achieve enhanced predictive accuracy.
4. **Recommendations:** The success of the Stacking Classifier encourages further exploration of ensemble techniques and model combinations. Future work may involve fine-tuning hyperparameters, experimenting with additional models, and incorporating advanced deep learning architectures for improved results.
5. **Continuous Improvement:** Machine learning models should be viewed as dynamic entities, subject to refinement and adaptation. Regular updates based on new data, changing patterns, or evolving requirements are essential for maintaining the relevance and accuracy of the models over time.

In conclusion, the study provides valuable insights into the performance of different models and ensemble techniques, serving as a foundation for future research and application in real-world scenarios.

The Future Work of this project is Prediction of Network Traffic and Work on the Security part of Network Slicing.

In the forthcoming research, the merging of predictive analytics with the advanced capabilities of 5G networks will be further explored, with a particular focus on the refinement of real-time traffic prediction and the enhancement of urban security measures. The utilization of 5G's high-speed data transmission and low latency will be leveraged to bolster predictive models, enabling dynamic adaptation to road conditions and ensuring swift responsiveness in traffic systems. Concurrently, the exploration of 5G's connectivity will facilitate the integration of a comprehensive surveillance network, where machine learning models will be employed to detect anomalies and strengthen urban security frameworks. An integrated approach will be pursued, enabling seamless information exchange between traffic and security prediction models, fostering an agile and interconnected urban ecosystem. Throughout this exploration, critical challenges such as data security and privacy concerns inherent in network transmissions will be prioritized for resolution. Collaborations with telecommunications entities, technology providers, and regulatory bodies will play a pivotal role in facilitating the successful integration and adaptation of the models to the evolving 5G landscape. Ultimately, the aim is to pioneer innovative advancements in the realm of smart and secure urban environments by harnessing cutting-edge connectivity and predictive analytics for network slicing in 5G

REFERENCES

[1] R. Li et al., "Deep Reinforcement Learning for Resource Management in Network Slicing," in IEEE Access, vol. 6, pp. 74429-74441, 2018, doi: 10.1109/ACCESS.2018.2881964.(RP -1)

[2] Y. Kim and H. Lim, "Multi-Agent Reinforcement Learning-Based Resource Management for End-to-End Network Slicing," in IEEE Access, vol. 9, pp. 56178-56190, 2021, doi: 10.1109/ACCESS.2021.3072435. (RP-5)

[3] M. Yan, G. Feng, J. Zhou, Y. Sun and Y. -C. Liang, "Intelligent Resource Scheduling for 5G Radio Access Network Slicing," in IEEE Transactions on Vehicular Technology, vol. 68, no. 8, pp. 7691-7703, Aug. 2019, doi: 10.1109/TVT.2019.2922668. (RP – 9)

[4] F. Song, J. Li, C. Ma, Y. Zhang, L. Shi and D. N. K. Jayakody, "Dynamic Virtual Resource Allocation for 5G and Beyond Network Slicing," in IEEE Open Journal of Vehicular Technology, vol. 1, pp. 215-226, 2020, doi: 10.1109/OJVT.2020.2990072. (RP – 4)

[5] A. Huang, Y. Li, Y. Xiao, X. Ge, S. Sun and H. -C. Chao, "Distributed Resource Allocation for Network Slicing of Bandwidth and Computational Resource," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9149296. (RP – 2)

[6] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han and C. S. Hong, "Network Slicing: Recent Advances, Taxonomy, Requirements, and Open Research Challenges," in IEEE Access, vol. 8, pp. 36009-36028, 2020, doi: 10.1109/ACCESS.2020.2975072. (RP – 3)

[7] A. Thantharate, R. Paropkari, V. Walunj, C. Beard and P. Kankariya, "Secure5G: A Deep Learning Framework Towards a Secure Network Slicing in 5G and Beyond," 2020 10th Annual PESU Confidential Page 13 of 14 HIGH LEVEL DESIGN DOCUMENT Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2020, pp. 0852-0857, doi: 10.1109/CCWC47524.2020.9031158. (RP – 6)

[8] S. Rathore, J. H. Park and H. Chang, "Deep Learning and Blockchain-Empowered Security Framework for Intelligent 5G-Enabled IoT," in IEEE Access, vol. 9, pp. 90075-90083, 2021, doi: 10.1109/ACCESS.2021.3077069. (RP – 7)

[9] Khan, S., Khan, S., Ali, Y. et al. Highly Accurate and Reliable Wireless Network Slicing in 5th Generation Networks: A Hybrid Deep Learning Approach. J NetwSyst Manage 30, 29 (2022).

[10] V. P. Kafle, P. Martinez-Julia and T. Miyazawa, "Automation of 5G Network Slice Control Functions with Machine Learning," in IEEE Communications Standards Magazine, vol. 3, no. 3, pp. 54-62, September 2019, doi: 10.1109/MCOMSTD.001.1900010. (RP – 8)

[11] X. Li et al., "Network Slicing for 5G: Challenges and Opportunities," in IEEE Internet Computing, vol. 21, no. 5, pp. 20-27, 2017, doi: 10.1109/MIC.2017.3481355.

Appendix A: Definitions, Acronyms and Abbreviations

Definitions:

- Machine learning: a type of artificial intelligence that allows systems to learn and improve from experience without being explicitly programmed
- Reinforcement learning: a type of machine learning where an agent learns to make decisions in an environment by maximizing a reward signal
- Deep learning: a subfield of machine learning that uses artificial neural networks to model and solve complex problems
- Transfer learning: a technique where a model trained on one task is re-purposed for another related task
- Supervised learning: a type of machine learning where a model is trained on labeled data to make predictions or classifications on new data
- Unsupervised learning: a type of machine learning where a model is trained on unlabeled data to find patterns or structure in the data
- Prioritized experience replay: a method used in reinforcement learning where important experiences are replayed more frequently to improve learning

Acronyms and Abbreviations:

- AI: Artificial Intelligence
- API: Application Programming Interface
- CPU: Central Processing Unit
- GPU: Graphics Processing Unit
- GUI: Graphical User Interface
- ML: Machine Learning
- RAM: Random Access Memory
- RL: Reinforcement Learning
- SQL: Structured Query Language
- UI: User Interface
- UX: User Experience

LIST OF FIGURES

LIST OF TABLES