UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Leonardo Azalim de Oliveira

User Equipment Traffic Classification in the 5G Core

Juiz de Fora
2025

Leonardo Azalim de Oliveira

User Equipment Traffic Classification in the 5G Core

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Edelberto Franco Silva

Coorientador: Prof. Dr. Luciano Jerez Chaves

Juiz de Fora

2025

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Oliveira, Leonardo Azalim de.
User Equipment Traffic Classification in the 5G Core / Leonardo Azalim de Oliveira. – 2025.
217 f. : il.

Orientador: Edelberto Franco Silva
Coorientador: Luciano Jerez Chaves
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2025.

1. 5G. 2. Tráfego de rede. 3. Classificação. I. Silva, Edelberto Franco, orient. II. Chaves, Luciano Jerez, coorient. III. Título.

Leonardo Azalim de Oliveira

User Equipment Traffic Classification in the 5G Core

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 11 de junho de 2025.

BANCA EXAMINADORA

Prof. Dr. Edelberto Franco Silva - Orientador

Universidade Federal de Juiz de Fora

Prof. Dr. Luciano Jerez Chaves - Coorientador

Universidade Federal de Juiz de Fora

Prof. Dr. Alex Borges Vieira

Universidade Federal de Juiz de Fora

Prof. Dr. Michele Nogueira Lima

Universidade Federal de Minas Gerais

Prof. Dr. Diogo Menezes Ferrazani Mattos

Juiz de Fora, 28/05/2025.

Documento assinado eletronicamente por **Alex Borges Vieira**, **Coordenador(a)**, em 24/07/2025, às 09:22, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Edelberto Franco Silva**, **Professor(a)**, em 24/07/2025, às 09:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Luciano Jerez Chaves**, **Professor(a)**, em 28/07/2025, às 13:18, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Diogo Menezes Ferrazani Mattos**, **Usuário Externo**, em 29/07/2025, às 14:39, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Michele Nogueira Lima**, **Usuário Externo**, em 29/07/2025, às 15:04, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **2422797** e o código CRC **69C1E830**.

To Ana Luce, whose support has always been with me through the hardest moments of my life.

ACKNOWLEDGMENT

My work would have been impossible to accomplish without the help of countless individuals, as achieving great things is rarely done alone.

During my Master's course, my family provided unwavering encouragement, which was vital to my success. I would like to express my gratitude to my parents Ana Luce and José Geraldo for always supporting me in my endeavors; to my sister Cecília for being the greatest example of resilience I know and a source of inspiration; and to my aunt Anayse for always believing in me and my work.

I also wish to thank my advisors, Professor Edelberto Franco Silva and Professor Luciano Jerez Chaves, for their invaluable guidance and partnership throughout my course.

The time I spent at the university allowed me to cultivate many friendships. I extend my thanks to Frederico Sales and Rodrigo Silva for everything they shared with me; to Khalid Usman for the memorable moments we shared during the course; to Yago Pereira for his assistance in revising the mathematical definitions used in this work; and to Rômulo Mello and his laboratory colleagues for their friendliness during our time in PGCC.

I am grateful to the professors of the Postgraduate program in Computer Science for their shared knowledge and expertise. I specially thank Professor Alex Borges Vieira for the invaluable lessons learned as a member of the Networks and Distributed Systems Laboratory (NetLab); Professor Mário Antônio Ribeiro Dantas for the enjoyable experience of working together while being his student; and Professor Marcelo Bernardes Vieira for the insights gained during his research methodology course.

I also received support from various individuals and organizations that enabled my research. I acknowledge the the Núcleo de Recursos Computacionais (NRC), coordinated by Professor Eduardo Pagani Julio for the technical support and equipment utilized during my research; the Centro de Gestão do Conhecimento Organizacional (CGCO), coordinated by Mrs. Patrícia Curvelo Rodrigues Stroele for the connectivity support during my defense; and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Rede Nacional de Pesquisa (RNP), Fundação de Apoio e Desenvolvimento ao Ensino, Pesquisa e Extensão (Fadepe), and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) for funding my research. I also thank the free5GC development team from National Yang Ming Chiao Tung University (NYCU) helpfulness during my collaboration on their project; and Mr Ali Güngör and all the contributors of the UERANSIM project for publicly releasing their state-of-the-art 5G UE simulator.

Finally, I would like to acknowledge the anonymous reviewers who evaluated the papers I published during my course, as well as the members of my defense committee for their insightful comments. I also extend my gratitude to every scientist who came before me and, in some way, laid the foundations that enabled my work to reach its current state.

"Science is an attempt, largely successful, to understand the world, to get a grip on things, to get hold of ourselves, to steer a safe course." (Carl Sagan, 1997, p. 29)

RESUMO

A quinta geração de redes celulares (5G), que é especificada pelo 3^{rd} *Generation Partnership Program* (3GPP), se distingue das gerações anteriores de redes móveis principalmente pela adoção de uma arquitetura baseada em serviços (do inglês, *Service Based Architecture* (SBA)). Ao aproveitar da flexibilidade desse novo paradigma arquitetural, este trabalho investiga a viabilidade da implementação de um mecanismo de *Network Data Analytics Function* (NWDAF) para a classificação de dispositivos 5G. Consoante com a normatização do 3GPP, o *International Telecommunication Union* (ITU) define três eixos de serviço 5G: *Enhanced Mobile Broadband* (eMBB), *Ultra Reliable and Low Latency Communications* (URLLC) e *Massive Machine-Type Communications* (mMTC). Esses eixos concentram principalmente aplicações que necessitam de altas taxas de vazão, baixa latência com alta disponibilidade e transmissões de baixo gasto energético com milhares de dispositivos de *Internet of Things* (IoT) conectados, respectivamente. Então, a classificação de diferentes tipos de dispositivos é crucial, uma vez que cada eixo do 5G está associado a restrições distintas, que influenciam a alocação de recursos no Núcleo 5G (do inglês, *5G Core* (5GC)). Definido pelo 3GPP na Release 15 como responsável pela análise de dados em redes 5G, o NWDAF permanece subexplorado na literatura. Então, para explorar essa lacuna, este trabalho utiliza um ambiente 5G simulado que emprega Software Livre (SL). Este ambiente engloba o UERANSIM, que é um simulador de *Radio Access Network* (RAN) e *User Equipment* (UE); o free5GC, que é uma implementação de 5GC; e um gerador de tráfego de rede customizável. Dois conjuntos de dados 5G reais, foram utilizados para treinar onze diferentes modelos de *Machine Learning* (ML) — *Linear Regression* (LR), *Histogram-based Gradient Boosting* (HGB), *Light Gradient Boosting Machine* (LGBM), *Multilayer Perceptron* (MLP), *Random Forest* (RF), *Linear Support Vector Classification* (SVC), *eXtreme Gradient Boosting* (XGB), *Decision Tree* (DT), AdaBoost, Stacking, e Voting — com o objetivo de classificar os UEs nos eixos a partir do tráfego de rede observado. No *pipeline* de ML implementado, os modelos alcançaram até 99.91% de F1-score para a classe eMBB ao utilizar RF, LGBM e XGB. Para a classe URLLC, os melhores resultados foram de 99.99% de F1-score com o SVC. Em contraste, devido à natureza esparsa dos dados da classe mMTC, enquanto o AdaBoost alcançou 99.99% de F1-score no cenário de *burst*, a performance ficou limitada em 2.74% no cenário probabilístico. Mesmo após empregar técnicas que incluíram o balanceamento dos dados de treinamento, os modelos sofreram de *overfitting*. Inspirado pelo movimento de ciência aberta, tanto o *software* utilizado quanto o conjunto de dados criado para a inferência estão acessíveis publicamente. Essa abordagem garante a reprodutibilidade e estabelece bases para investigações futuras sobre análise de dados conforme especificado pelo 3GPP.

Palavras-chave: 5G; tráfego de rede; classificação.

ABSTRACT

The fifth generation of cellular network (5G), specified by the 3^{rd} Generation Partnership Program (3GPP), distinguishes itself from previous mobile network generations primarily through the adoption of a Service Based Architecture (SBA). By leveraging the flexibility of this new architectural paradigm, this work investigates the feasibility of implementing a Network Data Analytics Function (NWDAF) mechanism for classifying 5G devices. Following the 3GPP specifications, the International Telecommunication Union (ITU) defines three 5G service axes: Enhanced Mobile Broadband (eMBB), Ultra Reliable and Low Latency Communications (URLLC), and Massive Machine-Type Communications (mMTC). These axes focus on applications that require high throughput, low latency with high availability, and low-energy transmissions with thousands of connected Internet of Things (IoT) devices, respectively. Therefore, the classification of different device types is crucial, as each 5G axis is associated with distinct constraints, which influence resource allocation in the 5G Core (5GC). Defined by the 3GPP in Release 15 as responsible for data analysis in 5G networks, the NWDAF remains considerably underexplored in existing literature. To address this gap, this work employs a simulated 5G environment utilizing Free/Libre and Open Source Software (FLOSS). This environment incorporates UERANSIM, a Radio Access Network (RAN) and User Equipment (UE) simulator; free5GC, a 5GC implementation; and a custom network traffic generator. Two real-world 5G datasets were used to train eleven different Machine Learning (ML) models — Linear Regression (LR), Histogram-based Gradient Boosting (HGB), Light Gradient Boosting Machine (LGBM), Multilayer Perceptron (MLP), Random Forest (RF), Linear Support Vector Classification (SVC), eXtreme Gradient Boosting (XGB), Decision Tree (DT), Adaboost, Stacking, and Voting — to classify UEs in the axes based on their observed network traffic. In the implemented ML pipeline the model inference performance results achieved up to 99.91% F1-score for the eMBB class using RF, LGBM, and XGB. For the URLLC class the best results were 99.99% F1-score with SVC. In contrast, due to the sparse data points of the mMTC class, while Adaboost achieved a 99.99% F1-score in the burst scenario, the performance was limited to 2.74% in the probabilistic scenario. Despite using techniques that include balancing the training dataset, the models suffered from overfitting. Inspired by the open science movement, both the software used and the dataset created for inference are publicly accessible. This approach ensures reproducibility and establishes a foundation for future investigations into data analysis as specified by 3GPP.

Keywords: 5G; network traffic; classification.

FIGURE LIST

TABLE LIST

ACRONYM LIST

1G	first generation of mobile telecommunications standards
2G	second generation of cellular network
3G	third generation of cellular network
3GPP	3^{rd} Generation Partnership Program
4G	fourth generation of cellular network
5G	fifth generation of cellular network
5GC	5G Core
5GS	5G System
6G	sixth generation of cellular network
ADRF	Analytics Data Repository Function
AF	Application Function
AI	Artificial Intelligence
AMF	Access and Mobility Function
AMPS	Advanced Mobile Phone System
AN	Access Network
AnLF	Analytics Logical Function
ARP	Address Resolution Protocol
AUC	Area Under the Curve
AUSF	Authentication Server Function
B5G	Beyond 5G
BS	Base Station
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CDR	Call Detail Record
CN	Core Network
CNN	Convolutional Neural Network
COTS	Commercial off-the-shelf
CP	Control Plane
CSV	Comma Separated Values
CUPS	Control and User Plane Separation
DAF	Data Analytics Function
DoS	Denial of Service
DN	Data Network
DNS	Domain Name System
DoH	DNS over HTTPS
DOI	Digital Object Identifier
DoT	DNS over TLS
DP	Data Plane
DT	Decision Tree
EDA	Exploratory Data Analysis
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core

FAD	free5GC Auto Deploy
FLOSS	Free/Libre and Open Source Software
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GNS3	Graphical Network Simulator 3
gNB	gNodeB
GPRS	General Packet Radio Service
GQUIC	Google Quick UDP Internet Connections
GSM	Global System for Mobile Communications
GTP	General Packet Radio Service Tunneling Protocol
GTP-C	GTP Control Plane
GTP-U	GTP User Plane
GTP'	GTP charging
HAC	Hierarchical Agglomerative Clustering
HD	High-definition
HGB	Histogram-based Gradient Boosting
HSPA	High Speed Packet Access
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
iid	independent and identically distributed
IMT	International Mobile Telecommunications
IoT	Internet of Things
IP	Internet Protocol
ISSN	International Standard Serial Number
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
kB	Kilobyte
KPI	Key Performance Indicator
KNN	K-Nearest Neighbors
LAN	Local Area Network
LGBM	Light Gradient Boosting Machine
LR	Linear Regression
LSTM	Long Short Term Memory
M2M	Machine to Machine
MANO	Management and Orchestration
MB	Megabyte
MCDM	Multiple-Criteria Decision Making
ML	Machine Learning
MLP	Multilayer Perceptron
MMS	Multimedia Messaging Service

mMTC	Massive Machine-Type Communications
MNO	Mobile Network Operator
ms	Millisecond
MTLF	Model Training Logical Function
MTU	Maximum Transfer Unit
NEF	Network Exposure Function
NF	Network Function
NFV	Network Function Virtualization
NGMN	Next Generation Mobile Networks Alliance
NMT	Nordic Mobile Telephone
NN	Neural Network
NR	New Radio
NRF	Network Repository Function
NS-3	Network Simulator 3
NSSF	Network Slice Selection Function
NTN	Non-Terrestrial Network
NWDAF	Network Data Analytics Function
OAM	Operations, Administration, and Maintenance
OCSP	Online Certificate Status Protocol
OS	Operating System
OvO	One-vs-One
PCAP	packet capture
PCF	Policy Control Function
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
PNIO	PROFINET IO
PoP	Publish or Perish
PPS	Packets per Second
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RAM	Random Access Memory
RAN	Radio Access Network
RAT	Radio Access Technology
RF	Random Forest
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RTT	Round-Trip Time
SA	Stand-Alone
SBA	Service Based Architecture
SBI	Service Based Interface
SDN	Software-Defined Networking

SLR	Systematic Literature Review
SMF	Session Management Function
SMS	Short Message Service
SSL	Secure Sockets Layer
SVM	Support Vector Machine
SVC	Support Vector Classification
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint Identifier
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
TPR	True Positive Rate
TR	Technical Report
TS	Technical Specification
TSG	Technical Specification Group
TTL	Time To Live
TV	Television
UAV	Unmanned Aerial Vehicle
UDM	Unified Data Management
UDP	User Datagram Protocol
UDR	Unified Data Repository
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UN	United Nations
UP	User Plane
UPF	User Plane Function
URL	Uniform Resource Locator
URLLC	Ultra Reliable and Low Latency Communications
VM	Virtual Machine
VoIP	Voice over Internet Protocol
WoS	Web of Science
XGB	eXtreme Gradient Boosting
XR	Extended Reality
ZSM	Zero-touch network and Service Management
ZTN	Zero-Touch Network

TABLE OF CONTENTS

1 INTRODUCTION

In the context of fifth generation of cellular network (5G), Brazil has reached over 20.5 million 5G connections as of 2023, with a total 40 million 5G connections registered in 2024 (2), while worldwide 1.6 billion 5G connections were established in 2023 (3). As the number of connected devices continues to rise, the efficiency and reliability of 5G networks become increasingly crucial to ensure seamless user experiences, support emerging use cases, and mitigate congestion and latency issues.

5G introduces a paradigm shift in mobile network architecture, leveraging a Service Based Architecture (SBA) for the first time. This approach marks a significant shift from previous generations, which were characterized by a more rigid and centralized reference-based architecture.

A SBA is based on a service-oriented concept, where components are designed to be coherent and loosely coupled. This enables the integration of diverse network elements and services. Network Function Virtualization (NFV) principles are also applied, where multiple entities from previous generations are virtualized as distinct Network Functions (NFs). Each NF is exposed through a standardized Service Based Interface (SBI) bus (4). Key benefits of the SBA include improved flexibility, scalability, and interoperability. The service-oriented approach enables seamless communication among network components.

The SBI bus and interfaces are part of the SBA design of the 5G Core (5GC). An 5GC is a conceptual element of the 5G network that contains most of the NFs and the main SBI bus. The 5GC architecture also adopts the concepts of Software-Defined Networking (SDN), which means that there is a separation between the Control Plane (CP) and the User Plane (UP) — that could be also referred as Data Plane (DP). This Control and User Plane Separation (CUPS) enables isolating the network traffic from the internal 5GC NF communication, thus allowing for the setup of multiple UPs controlled by the same CP NFs. Key functionalities of the CP NFs include UP packet processing management, policy configuration and enforcement, User Equipment (UE) charging, and general traffic monitoring, while the UP NFs forward user traffic (5, 6).

1.1 MOTIVATION

Experimenting with real equipment remains a costly endeavor. Therefore, specially when considering 5G and Beyond 5G (B5G) networks, simulated environments become critical for increasing efficiency and reducing the costs of experimentation and planning (7), even though these may have some limitations relative to real-world scenarios.

One approach to enhance the characteristics of 5G cellular networks is to leverage their NFs' flexibility and perform reconfigurations during run time. These reconfigurations may be automated and based on data collected within the network. According to the 3^{rd}

Generation Partnership Program (3GPP) specifications (8), a NF that is responsible for collecting data and providing insights is the Network Data Analytics Function (NWDAF).

As defined by 3GPP (9), the NWDAF can utilize Machine Learning (ML) models on data to automate analytics information generation, including NF load analytics. Furthermore, 3GPP specifies that the NWDAF may be employed in ML-assisted operations within the 5G System (5GS), which includes tasks such as Base Station (BS) frequency (10) or slice — an end-to-end logical network including network, computing, and storage resources — reconfiguration (11, 12), or even User Plane Function (UPF) selection as in (13). Consequently, evaluating model performance is essential for achieving accurate classifications in use cases focused on the correct grouping of UEs.

In this context, the 5G services can be separated in axes or vertical groups (14). According to (15), there are 3 main ways of classifying them: 3 axes by International Telecommunication Union (ITU), 5 axes by 3GPP and 14 axes by Next Generation Mobile Networks Alliance (NGMN). Considering the concepts and Key Performance Indicators (KPIs) presented by (15) and (16) on the 5G services, it is possible to deduce that the classification of each service to a given axis depends on the context and a given use case may fit in more than one group.

Thereby, the methodology that utilizes the network traffic to classify UEs in the service classes, presented in Chapter 4, was designed based on ITU's (16) 3-type classification: Enhanced Mobile Broadband (eMBB), Massive Machine-Type Communications (mMTC), and Ultra Reliable and Low Latency Communications (URLLC) defined in Subsection *2.1.3.1*. The classifier is intended to be integrated into an NWDAF instance, enabling the NWDAF to share classification results with any consumer NFs, as detailed in Subsection *2.1.3.3*.

1.2 RESEARCH GOAL

The main research problem is device classification based on observed traffic. Given the motivation presented in Subsection 1.1, the research goal is defined as follows: to create and use an open-source simulated 5G network environment to experiment with and assess the performance of employing ML models and data analysis techniques to classify UEs in the 5G service axes based on their network traffic, providing useful information for 5G Mobile Network Operators (MNOs) to support network operation decisions that improve reliability and efficiency.

1.3 MAIN CONTRIBUTIONS

The main scientific and technical contributions of this work are summarized below:

- Implementation and evaluation of ensemble learning methods for UE classification

Based on the findings from the literature review in Section 3.2, there is a notable absence of studies investigating the application of ensemble learning models for UE classification. Therefore, as outlined in Section 3.3, in addition to training ML models, the training of models based on ensemble methods was also conducted, with the results evaluated in Section 5.2.

- Generation of a 5G simulated traffic capture dataset

  Considering the absence of real world 5G publicly available datasets and some similarities shared by cellular and Wi-Fi (17), quite frequently, the work found in the literature utilizes Wi-Fi datasets for experimentation — e.g., as in (18, 19, 20, 21) — while another alternative approach involves the use of simulated data — e.g., (22, 23). This lack of 5G datasets is further supported by the literature mapping results discussed in Section 3.3 and the dataset survey described in Section 4.3. Therefore, the generation of new 5G traffic capture datasets is required, especially in the packet capture (PCAP) format. The created traffic dataset introduced in Subsection **5.1.1** was released in PCAP format, which is designed to store network traces (24, 25) and provides detailed packet information (e.g., flags and payloads), allowing for in-depth analysis, feature extraction, and traffic replay.

1.4 RESEARCH OUTPUT

Throughout the research conducted for this master's thesis, several papers were published at national and international conferences, a mini course was developed, and contributions to Free/Libre and Open Source Software (FLOSS) were made.

The short paper titled "Estudo e Avaliação de Métodos de Autenticação EAP na Infraestrutura de Redes de Telecomunicação 5G" (26) was presented at "XIII Workshop de Gestão de Identidades Digitais (WGID)" and published in September 2023 in the conference proceedings of "XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais". This paper includes a survey conducted to identify open-source tools that facilitate the simulation of 5G networks, leading to valuable experience with free5GC (27) and UERANSIM (28) — projects that would later contribute to the development of the free5GC Auto Deploy (FAD) tool (29) presented in Section 4.4 and integrate the environment presented in Section 4.2.

The paper titled "Análise da Funcionalidade da NWDAF no Core 5G Sobre um Conjunto de Dados" (30) was presented in the main track of "XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)" and published in May 2024 which holds a Qualis A4 rating. The findings from this work laid the foundation for the design and development of the ML pipeline detailed in Section 4.1, paving the way for practical applications of models, while representing the initial outcomes of this master's thesis.

A second paper that contributes to the same line of inquiry titled “A NWDAF Study Employing Machine Learning Models on a Simulated 5G Network Dataset” (31) was presented at “IV International Workshop on Distributed Intelligent Systems (DistInSys)” and published in June 2024 in the conference proceedings of “XXIX IEEE Symposium on Computers and Communications (ISCC)”. This paper refined the techniques employed in the previous work and highlighted the limitations of the previously created dataset, particularly regarding the quantity of packets. Notably, the experience gained was invaluable in implementing the NWDAF-based functionality detailed in Section 4.5 and the traffic generator and dataset outlined in Subsection **5.1.1**.

Another short paper titled “Eduroam e 5G: autenticação integrada via redes móveis e Wi-Fi no core 5G” (32) was presented at “XIV Workshop de Gestão de Identidades Digitais (WGID)” and published in September 2024 in the conference proceedings of “XXIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais”. This paper built upon the previous work and was significant for refining the 5G simulation environment, with the implementation of the FAD tool emerging as one of its outcomes, achieving a production-ready development status.

Additionally, the knowledge acquired and the details of the simulation environment, including the 5G network concepts underlying the FAD tool and its application in creating a testbed, were transformed into a mini course titled “Introdução a ambientes de experimentação 5G” (33), which was delivered during the “XXVI Semana da Computação do Departamento de Ciência da Computação da UFJF” in November 2024.

Part of the outputs of this work are related to supporting open science (34) in the computer networks field. Consequently, as part of the implementation efforts described in Sections 4.4 and 4.5, several code patches (35) and documentation enhancements (36) were submitted to the official repositories of free5GC and UERANSIM tools, being available to the community. Alongside the implemented patches, it was possible to actively participate in issue (37) and forum discussions (38) contributing to the improvement of the aforementioned tools. FAD, a tool to automate the deployment of the simulation environment presented in Section 4.2, was released under a FLOSS compatible license, being publicly available on GitHub (29).

Furthermore, the artifacts generated during the research are publicly available. The literature review records and the raw results of the conducted experiments (including model feature importance and model performance results – of the training, inference, and cross validation) and directory structure listing examples were published on Zenodo (39). The dataset utilized on the experimental phase was also published on Zenodo (40). The source code of the traffic generator and the NWDAF-based functionality implemented, including their documentation were also made available on GitHub (41).

After accumulating experience from the execution of the aforementioned works, this master's thesis refines the simulation environment and pipeline while contributing to advancements in the state of the art in UE classification, as highlighted in Section 1.3.

1.5 OUTLINE

The remainder of this master's thesis is organized into 5 chapters as follows: The theoretical background, including main concepts and specifications involved, is presented in Chapter 2. The literature review carried out during the investigation and the related work are outlined in Chapter 3. Chapter 4 details the design and implementation methodology of the 5G simulation environment and pipeline. Then, Chapter 5 contains analysis of the created dataset and the results of the experimental phase. Finally, Chapter 6 holds the conclusions and future work.

2 BACKGROUND

This chapter comprises the main concepts related to the research subject, including the roles of telecommunication standards organizations such as 3GPP and ITU, the evolving architectures of pre-5G and next-generation networks also including 5GC and related elements like the 5GS, and NFs such as UPF, ADRF, and NWDAF, as well as the GTP protocol and the 5G service axes. These concepts detailed in Section 2.1 are useful for the literature review and proposal in the subsequent Chapters 3 and 4. Additionally, the chapter presents in Section 2.2 the model life cycle terminology utilized and the evaluation of ML models in terms of performance metrics, thereby providing the foundation for the detailed analyses presented in Chapter 5.

2.1 MOBILE NETWORKS

The context of mobile networks encompasses various concepts. To clarify the essential definitions pertinent to this work, this section delineates the role of telecommunication standards organizations in 5G in Subsection **2.1.1**, provides an overview of the previous generations in Subsection **2.1.2**, discusses the 5G era in Subsection **2.1.3**, and examines the upcoming B5G networks in Subsection **2.1.4**.

2.1.1 Standards Organizations

The 3^{rd} Generation Partnership Program (3GPP) is a collaborative project between organizations that develop standards for cellular telecommunications. These "Organizational Partners" work together to create specifications — in the form of Technical Reports (TRs) and Technical Specifications (TSs) — that define 3GPP technologies encompassing radio access, core network, and service capabilities (42). A Release, the resulting set of specifications, provides a complete system description for mobile telecommunications. Currently, the Technical Specification Groups (TSGs) are: Radio Access Networks (RAN), Services & Systems Aspects (SA) and Core Network & Terminals (CT). Each TSG contains working groups related to its main subject and the TSG is responsible for coordinating and monitoring its working groups.

The 3GPP technologies are continually advancing through successive generations of commercial cellular and mobile systems, with fourth generation of cellular network (4G) and 5G marking significant milestones in this evolution. As the backbone for modern mobile systems beyond third generation of cellular network (3G), the 3GPP has emerged as the central hub for driving innovation and standardization in cellular technology, with its 4G and 5G work serving as a catalyst for widespread adoption.

Another important organization in the context of mobile networks is the ITU. The International Telecommunication Union (ITU) is a United Nations (UN) specialized agency

that promotes the development and harmonization of telecommunications technologies worldwide. With 194 member states, over 1,000 companies, universities, and international organizations as members, ITU facilitates global connectivity in communication networks by allocating spectrum, developing technical standards, and improving access to digital technologies in underserved communities. Headquartered in Geneva, Switzerland, ITU works to bring digital connectivity to everyone, providing a trusted platform for international agreements, knowledge sharing, capacity building, and collaboration with members and partners to promote universal connectivity and sustainable digital transformation (43).

While 3GPP focuses on developing technical specifications and standards for cellular networks — e.g., 3G, 4G, 5G, etc. — ITU comprises a broader range of telecommunication networks — e.g., broadband Internet, optical communications, etc. — and its standards include radio communication, satellite systems, network protocols, and multimedia coding. Notably, the two bodies quite often collaborate in the standardization processes (44): ITU sets the requirements and performance indicators for mobile networks — e.g., IMT-2020 (45) for 5G — while 3GPP develops the technical specifications to meet these requirements. Then, ITU reviews the respective 3GPP specifications to ensure global compatibility.

2.1.2 Previous Generations

Each generation of mobile network, which is surpassed at every decade or so (46), introduced new requirements but also new capabilities to the cellular communications. Table 1 outlines the main characteristics and example use cases of the different generations.

The first generation of mobile telecommunications standards (1G) was introduced in the 1980s. Characterized by analog audio transmissions, this era differed significantly from subsequent generations, which were fully digital.

During the 1G era, regional standards were developed and implemented in various countries, rather than a single global system. Notable examples include the Nordic Mobile Telephone (NMT) system and the Advanced Mobile Phone System (AMPS). The lack of a unified global standard resulted in a fragmented landscape, with different countries and regions utilizing distinct technologies for mobile communication. The limitations of analog systems led to their eventual replacement by the second generation of cellular network (2G) networks (47).

The 2G technology was launched globally in the early 1990s. A key difference between 2G and previous systems, retrospectively designated as 1G, is that radio signals are digital rather than analog for communication between mobile devices and BSs.

In addition to voice telephony, 2G enabled data services. The Global System for Mobile Communications (GSM) standard became the first globally adopted framework for mobile communications. The transition to digital technology enabled encryption for voice calls and data transmission, improving security while increasing capacity and

Table 1 – Summary of the characteristics of mobile networks generations

Generation	**Reference Decade**	**Main Characteristics and Use Cases**
1G	1980	Audio Signal Wireless Phone Calls
2G	1990	Voice Calls Multimedia Messaging Service (MMS) Pictures Basic Texting
3G	2000	Video Live Television (TV) Fast Internet Browsing
4G	2010	100 MBps speeds VoIP Calls High-definition (HD) Streaming Video Chat
5G	2020	100 × 4G speeds 4 GBps download speeds Internet of Things (IoT) Smart Cities Autonomous Cars Remote Robotics
B5G	2030+	Artificial Intelligence (AI) infused applications TeraHertz frequencies

Source: Created by the author (2025). Based on (46).

efficiency. 2G networks were primarily designed to support voice calls and Short Message Service (SMS). Later advancements, such as General Packet Radio Service (GPRS), enabled packet data services including MMS and limited internet access. 2G was succeeded by 3G technology, which provided higher data transfer rates and expanded mobile internet capabilities (48).

The 3G technology was rolled out in the early 2000s. The Universal Mobile Telecommunications System (UMTS) standard, created by the 3GPP, succeeded GSM.

When compared to the previous generation, 3G networks implemented significantly higher-speed mobile internet and enhanced multimedia capabilities, as well as improved voice quality. They provided moderate internet speeds suitable for general web browsing and multimedia content, such as video streaming. Later 3G releases, such as High Speed Packet Access (HSPA) and HSPA+, introduced improvements that enabled 3G networks to offer mobile broadband access with speeds up to 42 Mbit/s. The 3G was succeeded

by the 4G technology, which provided even higher data transfer rates and introduced advancements in network performance (49).

The fourth generation of cellular network (4G) was first adopted in the late 2000s and early 2010s. Compared to its predecessors, 4G has been designed to support all-Internet Protocol (IP) communications and broadband services, eliminating circuit switching in voice telephony and other 3G limitations (46).

The networks from 4G are not backward compatible with 3G due to significant differences in network architecture and technological advancements. It has enabled the adoption of Voice over Internet Protocol (VoIP) calls, video chat and real-time data exchange for the IoT, facilitating the growth of connected devices and smart systems. This generation has also expanded the availability of mobile TV and supports high-speeds up to 100 Mbit/s, and some low-latency applications that require Quality of Service (QoS) (46, 50). The 5G technology succeeded 4G, offering faster speeds, lower latency, and enhanced support for advanced use cases across multiple industries.

2.1.3 Fifth Generation of Cellular Network

The fifth generation of cellular network (5G) technology, first introduced in specifications in late 2017 (51), has been commercially deployed by MNOs worldwide since 2019 (52). There is one global unified standard for 5G: 5G New Radio (NR), which has been developed by 3GPP based on specifications defined by the ITU under the IMT-2020 (45) requirements. Compared to its predecessor, the 5G network significantly outperforms, offering substantially higher download speeds (up to 4 Gbit/s) and lower latency (e.g., achieving 1 ms Round-Trip Time (RTT) from the UE to the 5G BS, a notable improvement over 4G's 20 ms latency) (46).

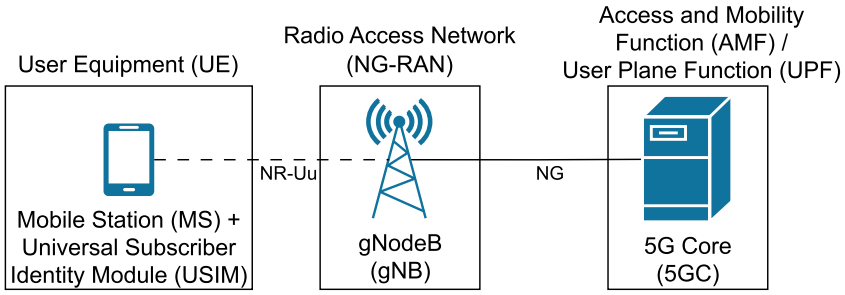
The increased bandwidth of 5G allows it to connect more devices simultaneously and improve the quality of cellular data services in crowded areas. This makes 5G well-suited for applications requiring real-time data exchange (such as Extended Reality (XR), autonomous vehicles, and remote surgery) (52) or big data applications (such as smart city environments and Machine to Machine (M2M) IoT communications) (46).

This generation has the potential of handling millions of IoT connected devices with stringent performance requirements and also extend beyond terrestrial infrastructure to include satellites and high-altitude platforms for global coverage through the 5G Non-Terrestrial Networks (NTNs). However, 5G deployment faces challenges such as significant infrastructure investment, spectrum allocation, and concerns about energy efficiency and environmental impact associated with the use of higher frequency bands (52).

From a computer networks perspective, the advent of 5G marked a significant departure from traditional cellular network architectures, pioneering the adoption of a service-oriented paradigm in its SBA, which diverged from the reference-based architecture

employed by 4G (52). The SBA is composed of multiple modular components (NFs) that implement and provide services as specified by 3GPP, enabling them to be consumed and integrated with other NFs through standardized SBIs, thereby facilitating greater flexibility, scalability, and interoperability (14).

Figure 1 – 5GS overview



Source: Created by the author (2025). Adapted from (14).

As illustrated in Figure 1, a 5G System (5GS) is a 3GPP-defined architecture that encompasses the 5G Access Network (AN), 5GC, and UE (53). It retains compatibility with most functionalities of 4G, including mobility between a 5GC and a 4G Evolved Packet Core (EPC), ensuring minimal impact on the user experience. The system employs familiar elements from previous generations with new names — UE, the NG-RAN featuring the gNodeB (gNB) as its primary node, and a 5GC — to deliver enhanced performance and flexibility compared to 4G.

2.1.3.1 5G Service Axes

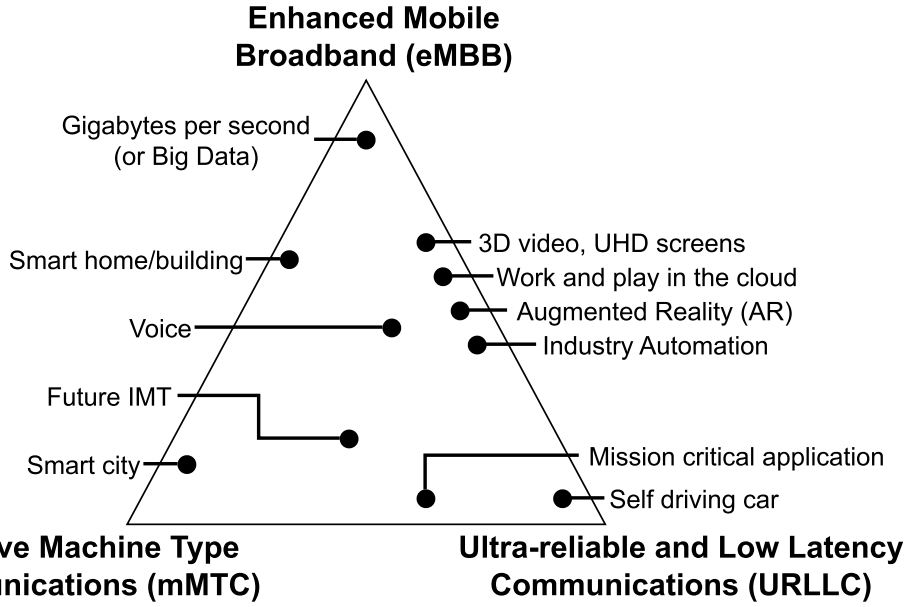
As 5G improves on the 4G services over several aspects (14), its services can be separated in axes or vertical groups. Considering the ITU's 3-type classification (16) for International Mobile Telecommunications (IMT), Figure 2 illustrates the axes and some usage scenarios.

The Enhanced Mobile Broadband (eMBB) axis is focused on providing high-capacity mobile broadband services for human-centric multimedia access. It encompasses a range of scenarios — from wide-area coverage with medium to high mobility and improved data rates compared to current capabilities, to hotspot situations that prioritize very high traffic capacity and elevated user data rates at the expense of mobility.

Then, Ultra Reliable and Low Latency Communications (URLLC) targets applications with stringent performance requirements. It is defined by its capabilities to deliver very high throughput, extremely low latency, and high availability, making it suitable for critical applications such as remote medical surgery, industrial control, smart grid automation, and transportation safety.

Finally, Massive Machine-Type Communications (mMTC) refers to a usage scenario involving a very large number of connected devices that typically transmit small volumes

Figure 2 – 5G service axes and usage scenarios of IMT for 2020 and beyond



Source: Created by the author (2025). Based on (16).

of non-delay-sensitive data. It emphasizes low device cost and extended battery life to support the proliferation of connected assets.

These axis definitions provided the foundation for selecting the applications utilized in the environment described in Section 4.2, as well as the class labels employed in the model performance evaluation presented in Subsection **5.2.3**.

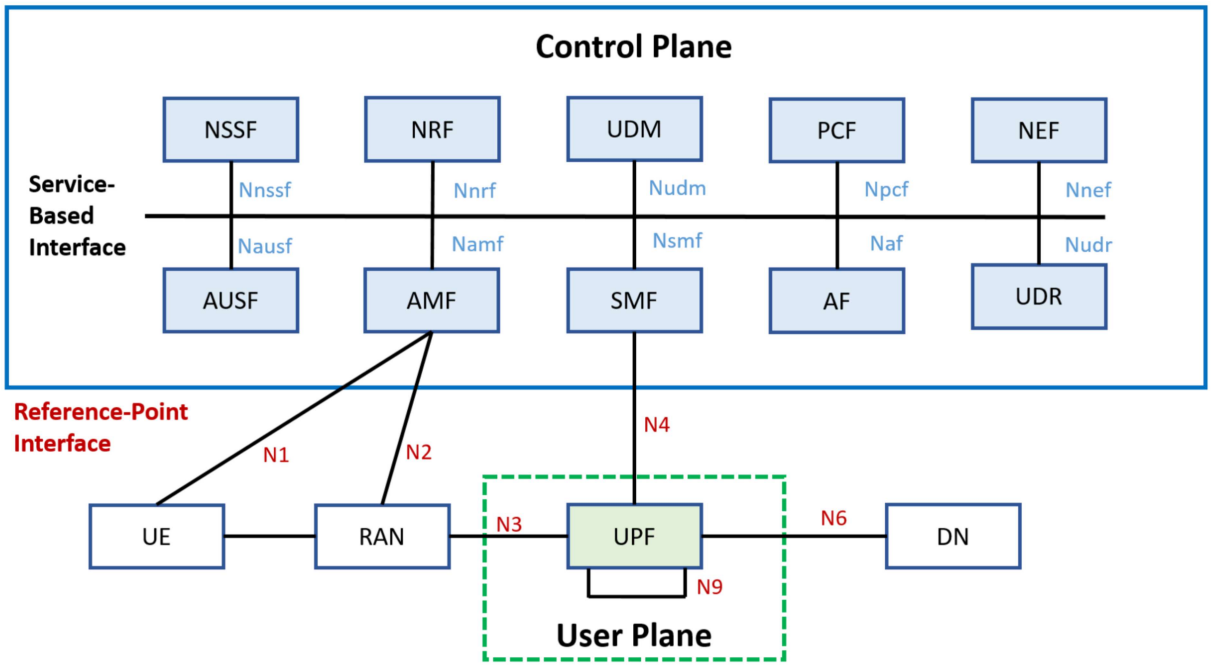
2.1.3.2 5G Core

The 5G Core (5GC) is a crucial component of 5G networks, comprising multiple NFs that can be divided into two SDN planes, forming the CUPS: the UP and the CP. This separation enables efficiency because, for instance, the network traffic from the UE can be routed directly from the Radio Access Network (RAN) to the UPF (54). Figure 3 provides an overview of the 5GC architecture, according to the CUPS paradigm. The NFs within the blue and green rectangles comprise the 5GC. It is designed using a SBA, wherein network functions provide modular and reusable services through standardized interfaces (depicted in blue), thereby enhancing agility and scalability in service provisioning (14).

From a logical perspective, the 5GC interfaces with the RAN via the `N2` interface, with the UE via the `N1` interface, and with the UPF via the `N4` interface (53). Further details regarding the UPF interfaces will be provided in Subsection *2.1.3.3*.

In a production environment, the RAN refers to a network infrastructure component consisting of radio BSs equipped with large antennas (55). Its primary function is to establish physical and logical connections between UEs and the 5GC (14). Conceptually (53), it may also be defied as a generic 5G AN. In contrast, the Radio Access Technology

Figure 3 – 5GS CUPS overview



Source: CHAI; LIN (54) (2021).

(RAT) pertains specifically to the physical layer communication protocols and standards employed in the RAN, which, in the context of 5G, corresponds to the NR standard.

A User Equipment (UE) can be defined as any equipment that allows a user to access network services. In the context of 5G (56), the interface between the UE and the network is the radio interface. Following the ITU definition in (57), a UE is a equipment that provides the functions necessary for the operation of the access protocols by the user. Examples of 5G UEs include smartphones, laptops, and any device capable of using a 5G wireless network interface to connect to a NR/gNB BS.

The General Packet Radio Service Tunneling Protocol (GTP) is an indispensable protocol of the 5G network architecture, enabling efficient communication between UE, gNB, and other NFs. As a tunneling protocol, GTP facilitates the delivery of GPRS packets within a mobile network, allowing for transmission over IP networks via User Datagram Protocol (UDP) as path protocol (58, 59).

The GTP protocol consists of GTP Control Plane (GTP-C), GTP User Plane (GTP-U), and GTP charging (GTP') components, each responsible for specific functions. The tunneling mechanism involves encapsulation of GPRS packets within a custom header structure, enabling efficient transmission. Key parameters in the GTP-U header include message type, packet length, sequence number, Tunnel Endpoint Identifier (TEID), and next extension header type. In 5G networks, GTP-U tunnels are employed to transmit user data between the gNB and the UPF (59).

The 3GPP specification (53) defines the role of User Plane Function (UPF) in

implementing the UP of the CUPS scheme, where it acts as a router for data traffic from and to the UE. The arrangement of the `N3`, `N4`, and `N6` interfaces, as depicted in Figure 3, facilitates direct connectivity between UE data and the Data Network (DN) (5, 6).

In more detail, the UPF encompasses a wide range of functionalities, including intra- and inter-RAT mobility anchoring, external Protocol Data Unit (PDU) session points of interconnect to DNs, packet routing and forwarding, inspection, policy rule enforcement, lawful intercept, traffic usage reporting, QoS handling, uplink traffic verification, transport-level packet marking, downlink buffering and notification triggering, end marker sending and forwarding, and response to Address Resolution Protocol (ARP) and IPv6 Neighbor Solicitation requests.

These functionalities are designed to ensure efficient and secure management of UP data traffic, while also supporting the requirements of various network slices and service scenarios. Notably, not all functionalities are necessarily supported in a single instance of the UPF, because the `N9` interface enables the deployment of multiple customized UPF instances within the same 5GC domain (14), allowing for flexibility and customization in accordance with specific network slice requirements.

Apart from the UPF, the specified (53) NFs for the 5GC include the Session Management Function (SMF), which oversees UPF selection, session management — e.g., between AN and UPF — IP address allocation, traffic steering, policy enforcement, and QoS; the Access and Mobility Function (AMF), responsible for mobility management, access authentication and authorization, security anchoring, and context management; the Policy Control Function (PCF), which establishes the policy framework governing network behavior and the CP NFs; the Authentication Server Function (AUSF) which provides authentication and authorization for UEs via the AMF and informs the authentication status to the Unified Data Management (UDM); the Network Repository Function (NRF), which facilitates the discovery of network function instances for inter-function communication; the Network Exposure Function (NEF), which supports third party independent functionalities — e.g., by exposing network function capabilities and events to Application Functions (AFs); the Network Slice Selection Function (NSSF), which selects the appropriate network slice instances and the optimal AMF for the UE; and the Unified Data Repository (UDR), which serves as the database for UE-related information — e.g., subscription data — complemented by the UDM, which supports the subscription management — e.g., the generation of authentication credentials — and acts as a front-end for the UDR.

2.1.3.3 Data Storage and Analytics

The Analytics Data Repository Function (ADRF) was introduced on Release 17 (60) and is specifically designed to provide a centralized repository for data or analytics. The ADRF enables consumer NFs to interact with its repository through functions that include

storing, retrieving, and removing data. These capabilities enable consumer NFs to leverage the ADRF as a trusted repository for their data. For instance, the NWDAF is expected to support the aforementioned key functionalities which means the ADRF can be used to support NWDAF operations, in particular while analyzing network performance, traffic patterns, and some relevant metrics, ultimately facilitating data-driven decision-making and optimized network operations.

The Network Data Analytics Function (NWDAF) is a component of 5G networks that collects, processes, and analyzes data from various sources to extract insights and trends, enabling informed decision-making and optimized network performance. As specified by 3GPP (61), it may gather data from UE, BS, or any other 5GC NFs, storing, processing, and analyzing it to identify patterns, anomalies, and trends. The NWDAF provides useful information on aspects such as NF load, network performance, observed service experience (e.g., Quality of Experience (QoE)), and UE-related analytics. It also typically integrates with other NFs like the UPF and the ADRF in a way that its analytics provide intelligence that can be leveraged to optimize network performance, enhance security, and increase operational efficiency.

The 3GPP Release 17 (9) specified that this NF could be functionally split in two other logical functions: Analytics Logical Function (AnLF) and Model Training Logical Function (MTLF) (62).

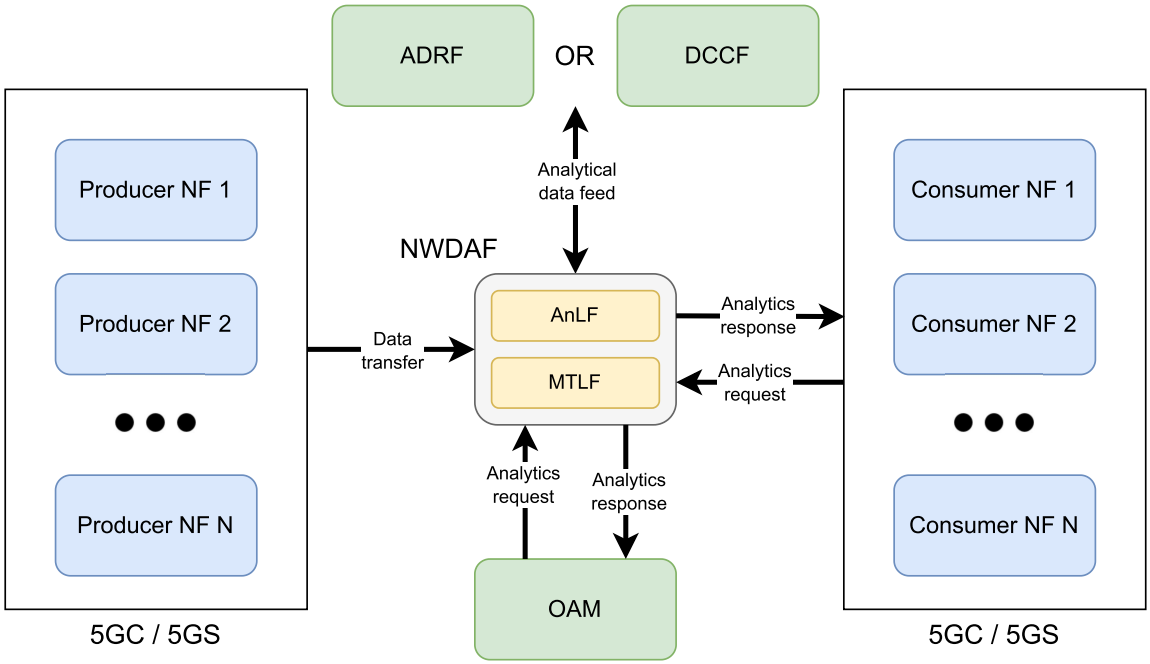
The AnLF is a component within the NWDAF tasked with processing and analyzing network data to yield insights. Its primary responsibilities encompass inference on collected data, statistical analysis of past events, and predictive analytics for future network behavior. The AnLF serves as the primary analytics engine, providing a gateway for analytics services through its service interfaces thereby facilitating the exchange of analytics and insights between various NFs and stakeholders. Its capabilities may be extend to the implementation with both statistical and ML models provided by the MTLF.

The MTLF is the other logical component of the NWDAF, which is specifically designed to train ML models utilizing collected network data. In essence, MTLF's primary functions revolve around the development and management of ML models. It serves as a enabler for other components within the 5GS by providing trained ML models to the AnLF or other NFs. This enables the seamless integration of ML-based analytics with other NFs of the 5GC (or even other NWDAF instances), thereby enhancing overall network performance measurement and decision-making capabilities with the timely update and provisioning of trained models.

Notably, the specification (63) allows analytics and collected data to be stored and retrieved from ADRF. As of Release 18 (61), trained models can also be stored on and retrieved from ADRF. Furthermore, UPF can serve as a data source for obtaining or calculating various metrics, including packet delay, bit rate, number of transmitted

packets, and packet retransmission rate. As discussed in Subsection **5.1.1**, the potential to utilize the UPF as a data source significantly influenced the decision to collect network traffic in this NF.

Figure 4 – Functional overview of the NWDAF



Source: Created by the author (2025). Based on (64) and (65).

Figure 4 provides an overview of how a NWDAF, integrating both AnLF and MTLF logical functions, interacts with other NFs. As specified in (61), a NWDAF integrated into the 5GC SBI bus facilitates CP interactions as illustrated in Figure 4. Producer NFs, such as the UPF, transfer data to the NWDAF, which can store and retrieve analytical data from the ADRF (if implemented), thereby establishing an analytical data feed. Consumer NFs, including the AMF and PCF, subscribe to receive information through analytics requests, which are answered with analytics responses that can support slice selection (11). Additionally, the Operations, Administration, and Maintenance (OAM) components, designed to enhance the management and monitoring of 5G transport networks, may utilize information from NWDAF to perform orchestration tasks, including BS frequency reconfiguration (10).

2.1.4 Beyond 5G

Future cellular networks are classified as Beyond 5G (B5G), representing an evolution built upon the foundation of 4G technologies (10). In addition, the definition of B5G encompasses both the evolution of 5G (such as 5G Advanced) and the emerging sixth generation of cellular network (6G) networks, which extend established 4G innovations while integrating new technologies to address the evolving demands of wireless communications (66, 67).

B5G/6G is expected to integrate AI at its core, enabling advanced air interfaces and network functionalities such as optimized symbol detection, channel estimation, and dynamic resource management (10). These networks will leverage on-demand and distributed machine learning for automated, intelligent operation, ensuring real-time responses with significantly reduced latency.

Furthermore, these systems will combine sensing and communication to enhance overall performance, reduce costs, and lower power consumption. Enhanced edge computing, efficient spectrum utilization, and novel security measures will support a diverse range of applications and services, positioning B5G/6G as an accumulative evolution over current technologies (68, 66, 67).

Although academia and standardization bodies such as 3GPP and ITU are already working on research developments and standards for B5G networks, 6G is notably still in its early stages (69).

2.2 MACHINE LEARNING

Machine Learning (ML) is a branch of the AI research field focused on the development and analysis of statistical algorithms that enable computer systems to learn from data and generalize to previously unseen data, thereby performing tasks without requiring explicit instructions (70). As explained in Subsection *2.1.3.3*, ML models may be used by 5G networks to perform analytics-related tasks. Therefore, it is important to define the pertinent terminology (Subsection **2.2.1**) and evaluation metrics (Subsection **2.2.2**).

2.2.1 Model Life Cycle Terminology

Machine Learning models undergo a structured life cycle that encompasses four main stages: experimentation, training, testing, and inference. It is essential to delineate these stages explicitly to ensure clarity and precision in the development and application of ML models.

The experimentation phase involves testing different algorithms and choosing which model architectures and training methods will be used within the life cycle. For instance, this could involve testing various models like Linear Regression (LR) or Decision Tree (DT) and exploring different hyperparameters to increase robustness (71).

Then, the supervised learning models training phase involves providing the selected ML algorithms with labeled or categorized data so it can iteratively process large, feature-rich datasets and recognize its patterns. Notably, the quality of the training dataset significantly influences model's performance (72).

It is crucial to evaluate an ML algorithm using a suitable dataset after training, accessing its performance by exposing it to previously unseen test data. This evaluation

process, known as data testing, entails comparing the model's output against actual results (also referred to as ground truth) for each example within the test set, thereby assessing the model's accuracy and reliability (72). It is worth mentioning that the results from this phase can be leveraged to improve model performance via hyperparameter optimization.

Following successful completion of the previous phases, models transition into the inference phase, where they are deployed to perform tasks without human intervention (71), such as classification tasks like object, image or packet classification.

2.2.2 Performance Metrics

In light of the potential variations in terminology and usage, which may lead to differing performance metrics for evaluating a ML model, this section explicitly defines each metric employed. The definitions displayed on Subsections *2.2.2.1* to *2.2.2.4* are based on (73) and (74). In classification problems, four key concepts are essential: True Positive (TP), where a correct prediction is made for a sample that belongs to the positive class; True Negative (TN), where a correct prediction is made for a sample that does not belong to the positive class; False Positive (FP), where an incorrect prediction is made for a sample that does not belong to the positive class; and False Negative (FN), where an incorrect prediction is made for a sample that belongs to the positive class.

Consider the multivariate Bernoulli variable definition from (75) and that a given n-th class C_n can be either positive/target (i.e., $C_n = 1$) or negative/background (i.e., $C_n = 0$). In multi-class classification problems, the target class is designated as the positive class, while the remaining classes are collectively regarded as the negative class. For example, let C_1 be the positive class and C_2 and C_3 the negative ones. Given a sample that belongs to the positive class (e.g., $C_1 = 1$) input to the classifier, if the output vector is $C_i = (1, 0, 0)$ then the result is correct and it will be considered as a TP, otherwise (e.g., $C_i = (0, 1, 0)$ or $C_i = (0, 0, 1)$) the result is not correct and it will be considered as a FN.

2.2.2.1 Accuracy

Let $\hat{y}_i$ be the predicted value of the i-th sample and y_i the corresponding true value, then the fraction of correct predictions over n_{samples} is defined in Equation 2.1, where $1(x)$ is called the indicator function.

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i) \tag{2.1}$$

2.2.2.2 Precision

Intuitively, the precision of a classifier is the ratio of TPs to all positive samples, i.e., it measures how well the classifier avoids labeling positive samples that are actually

negative. The precision can be calculated as defined in Equation 2.2.

$$\text{precision} = \frac{TP}{(TP + FP)} \tag{2.2}$$

2.2.2.3 Recall

Recall, intuitively, refers to the proportion of actual positive samples that are correctly identified by the classifier. In other words, it measures the classifier's ability to detect and retrieve all relevant instances of the positive class from the dataset. The recall reaches its optimal value at 1, while its lowest score of 0 signifies the worst case scenario. This value can be calculated using the formula defined in Equation 2.3.

$$\text{recall} = \frac{TP}{(TP + FN)} \tag{2.3}$$

2.2.2.4 F1-score

The F-measure can be interpreted as a weighted harmonic mean of the precision and recall. A F_β measure reaches its best value at 1 and its worst score at 0. With $\beta = 1$, F_β and F_1 are equivalent, and the recall and the precision are equally important as shown on Equation 2.4.

$$\text{F1} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.4}$$

2.3 SUMMARY

This chapter provides a clear overview of the foundational concepts essential to the research subject. It has defined the roles of key telecommunication organizations and discussed the architectural evolution across different generations of mobile networks — with an in-depth coverage of the 5G architecture. Furthermore, the chapter outlined the life cycle terminology and the performance metrics applied for evaluating ML models. This comprehensive background is important for the remainder of the master's thesis, providing the necessary context for the literature review in Chapter 3, the methodology presented in Chapter 4, and the analyses and discussions that will follow in Chapter 5.

3 LITERATURE REVIEW

This chapter provides an overview of the literature review conducted before the experimental phase of the research. Section 3.1 defines the literature review and delineates the methodology employed during the review process. The findings of the literature review were utilized in selecting the relevant works discussed in Section 3.2. The main points of the reviewed work are recapitulated in Section 3.3, while Section 3.4 offers a brief outline of the chapter's contents.

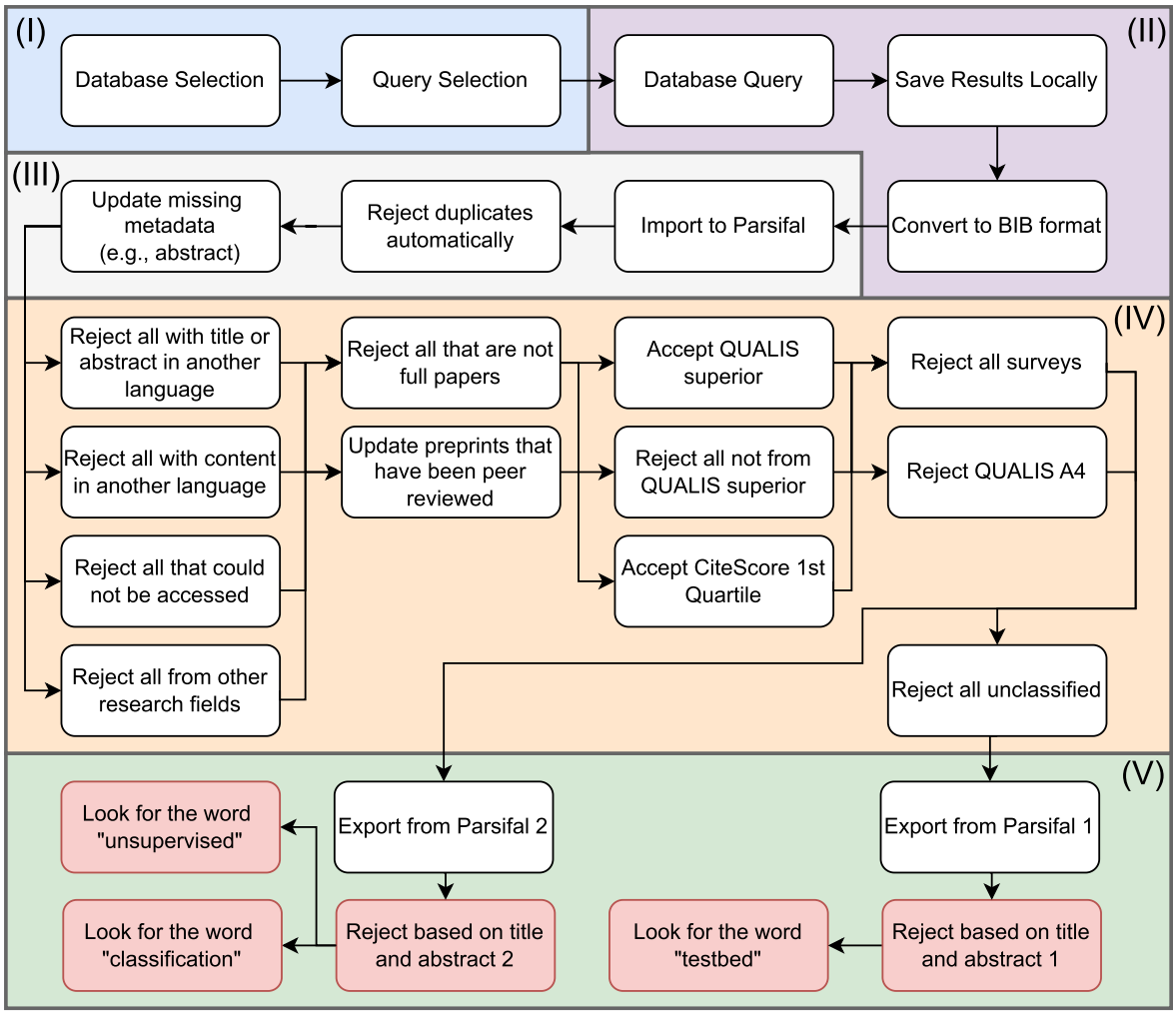
3.1 SYSTEMATIC LITERATURE REVIEW

A Systematic Literature Review (SLR) is "a systematic way of collecting, critically evaluating, integrating, and presenting findings from across multiple research studies on a research question or topic of interest." (78). Considering this definition, a literature review concerning the research topic was conducted. Figure 5 illustrates the main activities involved in the literature review, which will be elaborated upon in the subsequent paragraphs. Notably, it was not a complete secondary study or survey; rather, it was designed as a literature mapping aimed at investigating the relevant literature concerning this research topic. This approach serves to establish a theoretical foundation for this master's thesis and is structured to be reproducible, thereby facilitating its utility for other researchers interested in conducting SLRs.

According to (79), the selection of the databases is essential for the review process because the selection of a limited number of sources limits the literature explored. Following the three stages — Input, Processing and Output — of a literature review by (79) and the SLR methodology described on (80), the first two steps (present in the area I of Figure 5) consisted of identifying source databases and determining the query to apply.

For the purposes of this study, the selected databases comprise Scopus, Web of Science (WoS), Crossref, Google Scholar, Semantic Scholar, Springer Link, Science Direct and IEEE Xplore. The first two databases are recognized as reliable sources (80), while the next three are accessible via the Publish or Perish (PoP) software (81). The remaining databases were accessed through an institutional subscription. To enhance the breadth of literature reviewed, the search query utilized the keyword "NWDAF" and applied filters to exclude citations and patents where applicable. This query yielded 1,180 results across the selected databases, which were exported from the databases both manually or through the built-in export function of PoP (first two steps of the area II of Figure 5). The distribution of results from each database is illustrated in Figure 6. It is possible to observe that more than 71% of the records came from Google Scholar, while the other databases share between 5.5 and 1.4% of the records.

Figure 5 – Literature review main steps



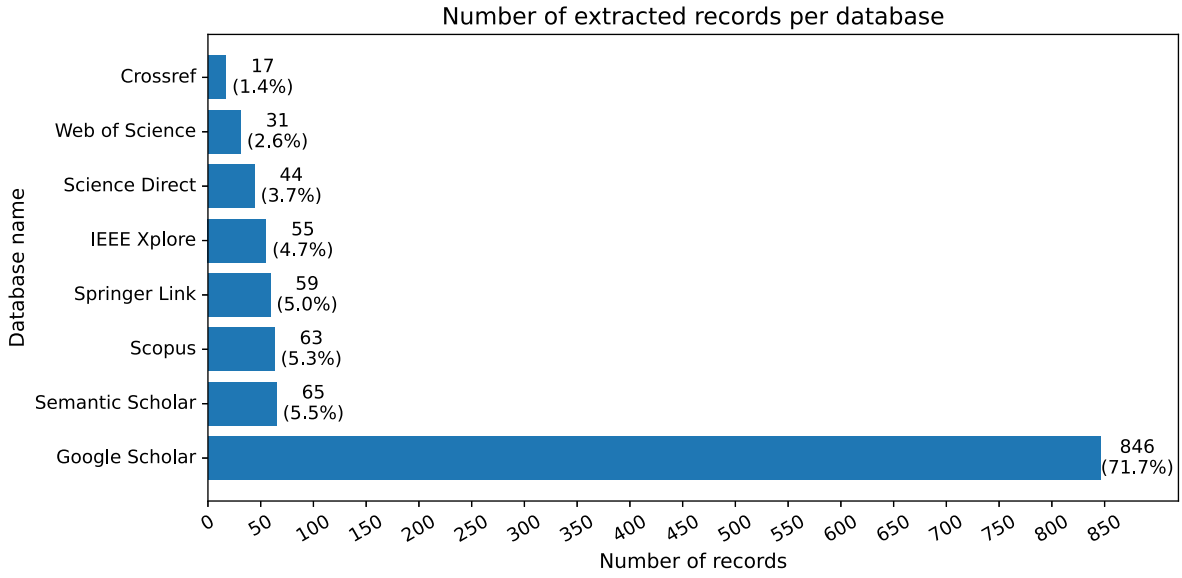
Source: Created by the author (2025).

The final step of the area II of Figure 5 was to convert the results to the widely used BibTeX (82) reference management format which is compatible with Parsifal (83), an online tool that was designed to help managing the considerable amount of data created during a SLR work more easily.

As represented in the area III of Figure 5, after loading the results to Parsifal, the first action taken was to use its built-in function to remove all the duplicate entries, which resulted in 834 entries left. Parsifal contains multiple metadata fields: title, abstract, year, author, keywords, author keywords, BibTex key, journal, document type, pages, volume, Digital Object Identifier (DOI), Uniform Resource Locator (URL), affiliation, publisher, International Standard Serial Number (ISSN), language and note. After removing the duplicates, all remaining entries were manually updated to add missing metadata (specially the abstract). While updating the ISSN, for the publications that had multiple entries the Electronic/Online ISSN had precedence over Print ISSN. After the metadata update, 10 new duplicate entries were removed, resulting in 824 entries left. One interesting finding

was that the majority of entries that required metadata update came from Springer. This possibly happened because of an extraction and/or format conversion issue.

Figure 6 – Extracted records per database



Source: Created by the author (2025).

The next step (shown in area IV) was to analyze the entries and reject those: with title or abstract in a language other than English or Portuguese; with content in a language other than English or Portuguese; that could not be accessed — e.g., published in databases not included in any subscription — and that belong to research fields other than Computer Science. This resulted, respectively, in 43, 3, 15, and 5 entries being rejected leaving 758 left.

Other steps performed were to update preprint records that have been already peer-reviewed and published to reflect these changes and to reject all entries that are not full peer-reviewed papers. This means rejecting books, book chapters, posters, citations, newsletters, technical reports, technical specifications, thesis, dissertations, short papers and preprints. These steps removed 222 entries.

The next steps included accepting some entries, which became candidates for further reading. Two evaluation systems were used: Qualis and CiteScore. Qualis (84) is the official scientific journal classification system maintained by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), a Brazilian federal government agency. Qualis ranks journals, proceedings, and annals according to a hierarchical strata, with A1 being the highest category (ranked from A1 to C). CiteScore (85) is another scientific evaluation system by Scopus/Elsevier which is based on impact factor metrics of journals and proceedings.

The entries that belong to a Qualis superior strata (that ranges from A1 to A4) or CiteScore 1st quartile were included and those not from Qualis superior (B1, B2 and C)

were rejected. On this step, 327 entries were included (315 by Qualis and 12 by CiteScore) and 38 were rejected.

Given the number of entries left — 327 — an extra exclusion step was performed. All 78 entries among the accepted ones containing a survey or review or proposal (theoretical only) or opinion paper and those 13 that belong to Qualis A4 strata were rejected. On this step (represented by the two blocks on the right side of area IV), Qualis had the least precedence order when an entry matched more than one criteria.

At this point, 236 papers remained for processing. The first approach was to reject all entries with "unclassified" status, and proceed to the result export (illustrated in Figure 5 on the right side of area V). The red-highlighted boxes in area V represent the subjective steps that rejected more entries based on the manual analysis of title and abstract. Using the keyword "testbed" it was possible to find 14 records.

Focusing on the classification techniques, a second run (represented on the left side of area V in Figure 5) utilized the keyword "classification" to find 9 papers. Given the low number of results on the second run, the keyword was also searched among the results rejected for the reason of not being included on any inclusion, then 4 extra papers could be found. Another paper was added to the included results by looking for the keyword "unsupervised" in this second export.

Table 2 – Literature review criteria used on Parsifal

Criteria	**Inclusion**	**Exclusion**
QUALIS superior A1	✓	
QUALIS superior A2	✓	
QUALIS superior A3	✓	
QUALIS superior A4	✓	
CiteScore 1st quartile	✓	
Is a survey		✓
Language other than English or Portuguese		✓
Not QUALIS superior		✓
Not a paper		✓
Not from Computer Science		✓
Paper could not be found		✓
QUALIS A4		✓
Title and abstract analysis	✓	✓

Source: Created by the author (2025).

The criteria used for the literature review were derived from examples of inclusion/exclusion criteria provided by (80). Table 2 contains an overview of the criteria used during the conducted review. It is essential to note that inclusion criteria "QUALIS superior A4" and exclusion criteria "QUALIS A4" refer to the same set of criteria in

different contexts as it was easier to differentiate them while using Parsifal's interface. According to the steps represented in Figure 5, the former was used as the criterion for record inclusion, while the latter was used as one criterion for exclusion on a later step. It is also important to note that the distinction between "Not a paper" and "Is a survey" is based on the fact that while surveys are a type of paper, it was decided to not include any surveys on the review because, as commented earlier, this work is neither a survey nor a complete secondary study.

Notably, after using Parsifal for processing the records throughout the literature review, the 1,180 extracted records and its corresponding metadata were archived and made publicly available on Zenodo (39) — a platform that supports the preservation of research outputs in any size or format (86) — allowing for future reuse.

3.2 RELATED WORK

From the 28 papers that resulted from the literature review described in Section 3.1, 12 were subjectively selected as related work after further reading.

The work in (87) targets QoE analysis and estimation for video playback. The authors propose a framework for automated data collection that incorporates user behavior simulation. It focuses on data available at application level and provides a public dataset of user interactions with the player, including abandonment, seeking, pausing, or no interaction. Therefore, facilitating a detailed analysis of KPI estimation models when evaluated on data derived from streaming sessions characterized by playback interactions. The authors affirm that model performance may be increased by an average of 42% if data with user interactions is added in the training of the models. Based on their findings, they conclude that user interaction could considerably affect ML model performance and that a small amount of data (e.g., if user early quits the playback) affects the model that monitors the utilized bandwidth.

In (88) the authors used Convolutional Neural Networks (CNNs) to classify packets collected from a 5G network RAN/gNB with the objective of detecting Denial of Service (DoS) attacks. The packets from the 5G-NIDD dataset (89) are processed and 9 features are extracted: Timestamp, Length, TTL, Highest layer, IP flags, Protocol, TCP features, UDP length and ICMP type. An initial feature analysis was performed and the features which are very similar or deterministic were removed. Windows of 10, 50 and 100 packets were created for each traffic flow. This data was used to create the matrices that form the images used as input to the CNNs. The proposed technique and a custom CNN achieved F1-Score results above 99.5%. The authors say that, according to the prediction times measured (below 50 ms), the technique could also be applied to real time classification scenarios. They also note that the NWDAF will play a more significant role in 6G networks particularly in intelligent threat mitigation. It is suggested that the NWDAF could be

instantiated on RANs, if a converged RAN-CN takes place on 6G. The key takeaway from the analysis is that, based on the results obtained, the proposed solution shows promise for deployment in upcoming 6G networks.

The authors of (90) develop their proposed framework's abstract data model around UE predictive classification. A long list of KPIs that could be monitored on the device, service and network is presented. The knowledge extracted through the framework called PRIMATE, was applied on network and service resource prediction for Hierarchical Agglomerative Clustering (HAC). This work used 4 different ML algorithms — Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Machine (SVM) — to compose a weighted voting model that classifies contextual information through predicting on which cluster those information would fit. After profiling the behavior of the devices using their contextual information and the behavioral clusters, each device is classified in one of the available profiles. The proposal, which was an extension of the NWDAF proposed by the 3GPP, was implemented on Network Simulator 3 (NS-3). As reported on the paper, the voting model performance achieved up to 96% of accuracy and 92% on F1-Score. As future work, it is proposed the integration of the proposal with some resource management system that may effectively use the output of the framework.

The work (91) performs anomaly classification using LR and eXtreme Gradient Boosting (XGB) ML algorithms on 5G synthetic data. Its experiment consists of a fixed network topology simulation that generates synthetic network traffic data that contain some fields as specified by 3GPP. The simulation contained 5 antennas and 5 types of devices (IoT device, vehicle, cell phone, smartwatch, and tablet computer) divided into 3 types of mobile data subscriptions (platinum, gold, and silver). The handover ratio was fixed during the entire simulation. The synthetic data was labeled and used to train the ML models in a supervised learning fashion. The ML mechanism was implemented using Python, but the source code was not shared. It is reported that the data used for feature extraction came from data rate, network area information, subscription categories and personal equipment ID fields and resulted in nine features. Considering all the cells and device categories, the average model performance for LR was 87% of Area Under the Curve (AUC) Receiver Operating Characteristic (ROC), 55.6% of accuracy, and 76.9% of precision. With slightly better results, XGB scored 90.7% of AUC ROC, 62.6% of accuracy, and 77.3% of precision. The conclusions indicate that it was possible to create a cell-based dataset for evaluating network data analytics in 5G, utilizing some information fields as defined by 3GPP. The authors future work include executing an Exploratory Data Analysis (EDA) in the created dataset to determine communication patterns, utilize the data fields specified by 3GPP as features, and perform a multiclass anomaly classification.

Focusing on the configuration and selection of 5G slices via the NSSF, the authors in (92) collected QoS and QoE-related data from UEs. Their proposal involved implementing an environment using the Graphical Network Simulator 3 (GNS3), which incorporated

a NWDAF to provide data to a proposed NSSF Data Analytics Function (DAF). The proposed NF employed a K-means ML model to cluster UE traffic flows, enabling Multiple-Criteria Decision Making (MCDM) methods to determine slice allocation weights on the 5GC. The authors concluded that the combined MCDM/ML approach is advantageous due to its simplicity and efficiency, particularly as it does not require historical data — an asset for newly created slices.

In (93), ML models were employed to classify cells and identify those with low performance based on expert domain knowledge. Using a Call Detail Record (CDR) dataset from an African MNO, the authors implemented a framework where K-means clustered the cells and a SVM classifier categorized these clusters. From the results of K-means clustering, four statistical measurements were calculated and visualized to facilitate expert analysis and labeling. Specifically, the average duration of normally terminated calls and the relative load of the maximum number of dropped calls were utilized to distinguish between cells with good and poor performance. The SVM classifier achieved an average accuracy of 97.69% in detecting cell anomalies. In summary, the proposed framework not only classifies cells based on performance metrics derived from hourly aggregated CDR data but also identifies performance-related issues within the network.

In (94), the authors implemented a network traffic classifier to detect voice and video streams, utilizing the Fraunhofer FOKUS Open5GCore (95) as the 5GC. The classifier processes data in 5-second windows to extract a packet length histogram, which is then normalized using a logarithmic function. Feature extraction relied on IF-ELSE structures, and five machine learning algorithms were evaluated and fine-tuned: SVM, Multilayer Perceptron (MLP), CNN, Long Short Term Memory (LSTM) and Recurrent Neural Network (RNN). The implemented function was integrated with a Management and Orchestration (MANO) system that controlled the 5GC's CP. The experimental setup included the 5GC, a simulated UE, and a concurrent traffic generator. The experiment involved capturing mixed traffic — including video playback from YouTube and Facebook Videos, as well as voice and video calls on Skype — for one hour. Time series analysis yielded an average accuracy of 94.9% for service classification. The results indicate that web traffic from YouTube and Facebook Videos exhibited high similarity, and the authors noted that the focus on voice and video streams is justified given their substantial resource demands on a 5G network.

An implementation focused on detecting IoT botnet attacks on 5GC was presented in (59). The experimental setup included a simulated UE and the Fraunhofer FOKUS Open5GCore (95). The methodology involved replaying MedBIoT dataset (96) packets to generate 5G traffic, with PCAP files exported in raw format for subsequent feature selection. Random sampling and feature encoding (a form of normalization) were applied to the data, and K-fold cross-validation was used to mitigate overfitting. Four ML algorithms were employed — KNN, SVM, RF, and a stacking ensemble meta-model — performing

both binary and multiclass classifications among benign traffic and three types of botnets. Notably, the models heavily relied on IP address-related features, consistently ranking among the top three most significant features. Performance was evaluated using metrics such as accuracy, precision, recall, F1-score, and AUC ROC. In an environment with a 50% benign-to-malicious packet ratio using approximately 40,000 packets, the stacking ensemble algorithm achieved the best results, reaching 99.96% accuracy for binary classification and 98.72% for multiclass classification. The authors concluded that detecting IoT botnets in the 5GC environment using only GTP-U packets is challenging, and that IP packet features play a crucial role in accurate classification. They also suggest that the implemented scheme could be valuable to future NWDAF deployments, specially in 6G networks.

Using the NWDAF architecture as a reference, the work presented in (97) implemented a ML classifier to differentiate device types based on their network traffic. The experiment was conducted in a simulated 5G environment employing free5GC as the 5GC and UERANSIM as the RAN simulator. ML functionality was developed in MATLAB (98) with parameter optimization performed; however, resource allocation was not addressed and the source code remains unavailable. The features derived from a default Wireshark (99) capture included traffic time, source and destination addresses, protocol name, packet length, and packet information. Notably, protocol names were excluded during training as they were used as the classification output. Fourteen ML models were evaluated, including Fine, Medium, Coarse, and Optimizable Trees; Linear, Quadratic, and Boosted Trees SVMs; Ensemble Bagged Trees; Ensemble RUSBoosted Trees; and Medium, Wide, Bilayered, and Trilayered Neural Networks (NNs). The best accuracy, exceeding 95%, was achieved by models such as Fine Tree, Medium Trees, Optimizable Trees, SVM Boosted Trees, Ensemble Bagged Trees, Medium NN, and Wide NN. Medium Trees also registered the fastest training time at 11.6 seconds. The authors conclude that it is feasible to generate traffic and perform service classification solely from UE network traffic. However, further investigation is needed on methods to assist the 5GC in correctly allocating users to slices and adjusting these slices as user behavior evolves. It is also commented that resource allocation, slice management, and mobility management in cellular networks remain open research challenges.

The authors of the work (100) propose a method to classify device types by analyzing encrypted traffic behavior through statistical analysis of packet header data without relying on payload information or protocol-specific packets. A testbed consisting of 39 IoT devices, 3 non-IoT devices, and a network router was utilized to generate data that was mixed with another dataset. In the first stage of the method, a feature dataset was generated by extracting the number of packets, total packet lengths, average packet lengths, and the number of destination addresses per device over 10 minutes. In the second stage, two datasets with 6,000 features were created: one containing the number of packets sent per device every 100 ms over 10 minutes, and another with the sum of

packet lengths per device in the same interval. For the training phase of the first stage (classifying IoT, non-IoT, and router), the performance metrics achieved up to 97.6% accuracy and 92.4% F1-score for RF. For the testing phase of the second stage (multiclass classification per device type), the best result of accuracy was 82.6% and 86.8% for the LSTM+CNN model on the two datasets tested. The dataset contained 141,654 total packets with class imbalance. The authors conclude that the proposed method effectively classifies IoT devices by analyzing packet header statistics in the first stage. In the second stage, the system was able to classify a wider range of IoT device types, with the combined LSTM+CNN model achieving 12% higher accuracy with a training time that was 40 times shorter than ResNet 1D model. The method is expected to facilitate traffic classification by device type for capacity planning in networks with a massive number of dynamically connected devices.

In (101), a NWDAF prototype compatible with Open5GS (102) was implemented to monitor NF traffic within the 5GC and assist MANO systems. Although the source code is unavailable, the dataset — comprising 138 minutes of captured traffic and 171,821 packets — was shared. The environment used Open5GS as the 5GC and UERANSIM as the UE simulator. To develop the models, seven features were extracted from each packet: No., Time, Source IP, Destination IP, Protocol, Length, and Info; packets not belonging to inter-NF traffic were excluded. A K-means clustering algorithm was then applied to group NFs based on source and destination NF, average and maximum packet length, standard deviation of packet length, and total number of transmitted packets for each NF pair. Through clustering and unsupervised learning, the authors demonstrate the ability to identify similarities in NF interactions and apply NWDAF insights to Intelligent Networks and NFV MANO, highlighting NWDAF's potential to support efficient network management in B5G networks. The authors emphasize that NWDAF is crucial for intelligent networks and MANO tasks, assisting in resource allocation and anomaly detection.

Finally, a conditional packet classifier was implemented to classify 5G traffic by service in (103). The traffic generation environment comprised free5GC as the 5GC and three UERANSIM instances. Without relying on ML models, the author implemented three conditional techniques: The first technique evaluates whether the protocol is Transmission Control Protocol (TCP); if so, it analyzes the payload size, assigning small payloads to mMTC and larger ones to eMBB, while UDP packets are classified as URLLC. The second technique matches the source or destination port against a predefined service list for classification; if no match is found, the first technique is applied. The third technique assigns 18 weights (six per class) to various parameters, with the classification result determined by the highest total score. As reported in the work, the best performance was achieved with the third technique, which reported an average accuracy of 99.8% (per class: 100% for eMBB, 79.15% for mMTC, and 99.84% for URLLC). The source code

was provided as code listings, and the approach offers a greater level of explainability compared to black-box ML models. The packet dataset was heavily imbalanced (5.15% eMBB, 0.45% mMTC, and 94.40% URLLC), likely due to the characteristics of the traffic streams. On the one hand, URLLC traffic contained a capture of a remote-controlled Unmanned Aerial Vehicle (UAV) coupled with a camera, thus generating a high number of packets. On the other hand, eMBB and mMTC traffic were respectively generated by a smartphone browsing web pages and an IoT sensor sending periodic data. This study offers an analysis of traffic classification in 5G networks, demonstrating the effective use of simulators such as free5GC and UERANSIM to emulate network topologies and behaviors. It also discusses the advantages and limitations of using high-level languages such as Python for packet classification, with particular emphasis on practical considerations for QoS-critical applications. Future work includes applying ML models to enhance the weighting approach used in the third technique.

3.3 COMPARISON

Research efforts are underway to explore how the NWDAF can be used on current 5G networks (62) and serve as an enabler for future closed-loop network architectures without human intervention (104), including Zero-Touch Network (ZTN) and Zero-touch network and Service Management (ZSM) networks (93, 105). These advancements represent a crucial step towards the development of proactive networks driven by AI (88), which will contribute to the emergence of the 6G. Within this context, the investigation of the state of the art is an important step that serves as a reference to direct the current research.

The 12 studies listed in this master's thesis have contributed to the advancement of 5G network traffic classification and related areas, frequently utilizing ML models. For instance, in (87) the effect of user interaction in the classification performance of models is investigated. Moreover, (59) and (88) focus on enhancing the security of 5G networks by, detecting DoS and IoT botnet attacks, respectively. A prototype for the NWDAF to monitor NFs traffic within the 5GC is proposed in (101), which can address anomaly issues, alongside UE anomaly classification discussed in (91). The NWDAF serves as a data source for the configuration and selection of 5G slices, as proposed by (92). Additionally, (93) emphasizes the classification of cells and the identification of those with low performance. The development of a network traffic classifier for detecting voice and video streams is presented in (94), while (90) explores UE predictive classification into profiles. Furthermore, (97) introduces a ML-based classifier designed to differentiate device types based on their network traffic, and (100) examines the classification of device types by analyzing encrypted traffic behavior through statistical analysis. Finally, (103) presents a conditional packet classifier aimed at classifying 5G traffic by service without relying on ML algorithms.

Table 3 delineates the key characteristics of each reference cited in this chapter, with an emphasis on ML aspects. It should be noted that, with the exception of (103), the references were obtained through the literature review described in Section 3.1. In contrast to the available literature, (103) proposes a ML-independent and straightforward yet efficient approach to traffic classification that utilizes only conditionals for classifying network packets.

As elaborated in the following Chapter 4, the approach taken in this study aligns with several existing studies in the literature. Similar to the work presented in (94), IF-ELSE structures were employed for feature extraction, with YouTube utilized as a traffic source. Features were extracted from PCAP files, in line with the methodology in (59). Additionally, feature encoding and K-fold cross validation were implemented. Based on the findings in (59) and prior experience (31), IP addresses were excluded from the feature set during ML model training. In contrast to (97), resource allocation is not involved in the current implementation. Similar to (100), the classification technique described in Chapter 4 does not rely on payload data, and class imbalance within the dataset was faced. Furthermore, protocol-specific packets were incorporated without aggregating them into windows.

Table 3 – Comparison of classification techniques

Reference	**ML models**	**ML performance metrics**	**Number of features**	**Extracted features**
(87)	DT RF	Accuracy Precision Recall	10	pckt_count pckt_size throughput iat pckt_size_gt100 pckt_count_gt100
(88)	Custom-CNN ResNet-V2 MobileNet-V2 EfficientNet-V2S DenseNet-201 Inception-V3 Xception	F1-score	9	Timestamp Length TTL Highest layer IP flags Protocol TCP features UDP length ICMP type
(90)	DT RF KNN SVM	Accuracy F1-score	3	Packet Size(b) Downlink Delay(s)

Table 3 (continued) – Comparison of classification techniques

Reference	**ML models**	**ML performance metrics**	**Number of features**	**Extracted features**
(91)	LR XGB	Accuracy Precision AUC ROC	9	last2_mean last4_mean last8_mean per_change_last2 per_change_last3 per_change_last4 change_last2 change_last3 change_last4
(92)	K-means (to help multicriteria methods)	RMSE (clustering)	9	Latency Jitter Loss Bandwidth (Mbps) Transfer UE Experiment (id) Distance Reliability
(93)	Custom SVM	Accuracy	7 (cell related)	Date Time Traffic Volume Call Count Number of SMS
(94)	SVM MLP CNN LSTM RNN	Accuracy	6	Packet Length (6 range functions)

Table 3 (continued) – Comparison of classification techniques

Reference	ML models	ML performance metrics	Number of features	Extracted features
(59)	KNN SVM RF meta-model (stacking ensemble)	Accuracy Precision Recall F1-score AUC ROC	18 10 28 (experiment dependent)	eth.src eth.dst arp.src.proto ipv4 arp.dst.proto ipv4 arp.src.hw mac arp.dst.hw mac arp.opcode ip.src ip.dst ipv6.src ipv6.dst icmp.type icmp.code tcp.srcport tcp.dstport udp.srcport udp.dstport frame.len gtp.flags gtp.message gtp.length gtp.teid gtp.ext_hdr.next

Table 3 (continued) – Comparison of classification techniques

Reference	ML models	ML performance metrics	Number of features	Extracted features
(97)	Tree-based: Fine Medium Coarse Optimizable SVM-based: Linear Quadratic Boosted Trees Ensemble-based: Bagged Trees RUSBoosted Trees NN-based: Medium Wide Bilayered Trilayered	Accuracy	6	traffic time source IP destination IP protocol name packet length packet information
(100)	1st stage: LR RF SVM 2nd stage: MLP LSTM+CNN ResNet 1D	Accuracy F1-score	4	# packets sum packet lengths avg packet lengths # dst addresses
(101)	K-means (to cluster traffic)	N/A	7	No. Time Source Destination Protocol Length Info
(103)	N/A	N/A	N/A	N/A

Table 3 (continued) – Comparison of classification techniques

Reference	ML models	ML performance metrics	Number of features	Extracted features
Ours	LR HGB RF DT MLP Linear SVM LGBM XGB AdaBoost Stacking Ensemble Voting Ensemble	Accuracy Precision Recall F1-score AUC ROC Score	33	Packet_no Timestamp Time_delta Source_IP Destination_IP Frame_type Frame_total_length Frame_header_length Frame_payload_length Source_port Destination_port TCP_completeness TCP_compl_reset TCP_compl_fin TCP_compl_data TCP_compl_ack TCP_compl_syn_ack TCP_compl_syn TCP_compl_str TCP_flags_bin TCP_flags_str TCP_window_size TCP_window_size_scale Frame_protocols IP_protocols IP_flag_reserved_bit IP_flag_dont_fragment IP_flag_more_fragments TTL TCP_header_length Data_length QUIC_packet_length QUIC_length

Source: Created by the author (2025).

In the context of open science characteristics, Table 4 compares the works cited in this chapter. Among the references examined, only five (59, 87, 91, 88, 101) utilized or released public datasets, with just three available in PCAP format. Notably, only one work (88) fully released the implemented source code.

Table 4 – Comparison of open science characteristics

Reference	**Year**	**Dataset is Public**	**Source Code is Public**	**Aim**
(91)	2020	✓ (csv)		Anomaly detection
(93)	2021		Pseudocode	Cell classification
(94)	2021			Voice and video stream detection
(59)	2022	✓		Attack detection
(87)	2022	✓ (csv)		User interaction detection
(90)	2022			UE classification
(92)	2022		Pseudocode	Slice selection
(101)	2022	✓		Anomaly detection
(103)	2022		Listings	UE classification
(97)	2023			UE classification
(88)	2024	✓	✓	Attack detection
(100)	2024			UE classification
Ours	2025	✓	✓	UE classification

Source: Created by the author (2025).

Consequently, a network traffic dataset was created following the method in (101), as detailed in Subsection **5.1.1**. In contrast to the references focused on classifying UEs, this work not only released the utilized dataset but also provided full access to the source code, as elaborated in Sections 4.4 and 4.5. Unlike reference (97), which employed MATLAB, the code was published using the FLOSS programming languages Python and Bash. Furthermore, consistent with (97), (101), and (103), the experimental setup utilized the free5GC and UERANSIM open-source projects, as described in Subsection **5.1.1**.

Building on the results found on the literature, which are characterized in Tables 3 and 4, the experiments conducted in this study, outlined in Chapter 4, include the inference phase that tests the models on unseen data, as detailed in Section 4.1. The 5G simulated environment (discussed in Section 4.2) was implemented using FAD, a FLOSS tool (described in Section 4.4) that automates the delivery of the software stack necessary to execute the packet capture experiment executed to create the dataset presented in Subsection **5.1.1**. Additionally, as highlighted in Table 3, the evaluation of eleven supervised learning models, including AdaBoost and two custom ensemble learning models (Stacking and Voting) identified in the literature is performed, with results contained in Subsection **5.2.3**. Supporting the open science characteristics verified in Table 4, the synthetic 5G network traffic capture dataset (40) created for this study was released in PCAP format, facilitating further experimentation and analysis. The source code is also available (29, 41) enabling future study and reuse.

3.4 SUMMARY

This chapter delineated the methodology employed during the literature review process, detailed the relevant work, and encapsulated the main points discussed in the reviewed literature. To improve the reproducibility and reuse of the literature mapping results, the extracted records and its corresponding metadata are accessible on Zenodo (39). Following a comprehensive review, twelve papers were identified as pertinent to this study, and a comparative analysis of these works and the current research is presented from two distinct perspectives: ML and open science characteristics, as indicated in Tables 3 and 4.

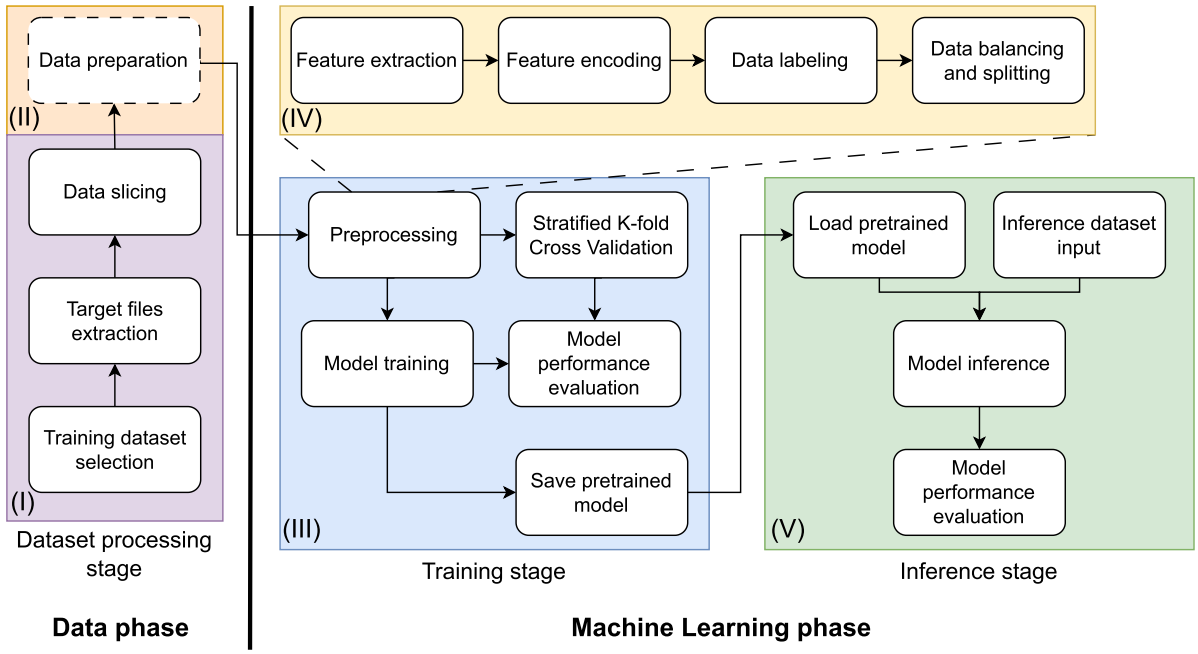
4 METHODOLOGY

The methodology outlined in this chapter is organized into two primary components: the ML pipeline, presented in Section 4.1, and the simulation environment, which will be detailed in Section 4.2. The datasets utilized during the training phase of the pipeline are introduced in Section 4.3. Section 4.4 introduces free5GC Auto Deploy, the FLOSS tool that facilitates the installation and configuration of the 5G simulation environment. The steps involved in implementing the NWDAF-based functionality are detailed in Section 4.5.

4.1 ML PIPELINE

In accordance with the NWDAF definition and specification outlined in Subsection *2.1.3.3*, a ML pipeline was developed to facilitate the study and effective utilization of ML techniques and network data within the context of 5G. This pipeline, illustrated in Figure 7, is divided into three primary stages: dataset processing, training, and inference.

Figure 7 – Outline of the designed ML pipeline



Source: Created by the author (2025).

The first stage, dataset processing (depicted in area I), is initiated with the selection of datasets that will be utilized for model training in the subsequent phase. It may be performed manually by domain specialists to ensure relevance and alignment with the training objectives. An example of datasets suitable for selection in this step is presented in Section 4.3. Following this selection, relevant files that align with the training objectives are extracted from the datasets. The data slicing step is then performed to identify packet flows of interest and/or to reduce the volume of packets available for the next phase. If

the data requires additional formatting or processing for proper ingestion by the models during the ML phase, an optional data preparation step (area II) may be performed. An example of this implementation will be provided in Subsection **4.5.1**.

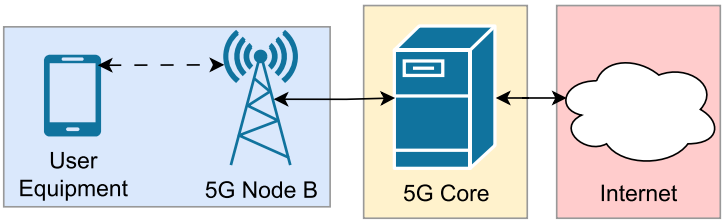
Upon completion of dataset processing, the training phase (illustrated in area III) initiates. This phase begins with data preprocessing (represented in area IV), which encompasses feature extraction, encoding, data labeling, and data balancing and splitting to create distinct training and test data splits, thereby preparing the data for model training. Once preprocessing is complete, the actual model training takes place, accompanied by cross validation to evaluate model robustness. The trained models are subsequently saved for future use, and a performance evaluation is conducted to assess the results of the training phase, which will be outlined in Subsection *5.2.3.1*.

Finally, the inference phase (depicted in area V) involves loading the pretrained models and the inference data to execute the inference step. The experiment conducted to create the dataset for this phase during the execution of this master's thesis is detailed in Subsection **5.1.1**. A performance evaluation is performed once again to assess the inference results, which will be presented in Subsection *5.2.3.2*.

4.2 5G SIMULATION ENVIRONMENT

Building on the findings in (7) and from the literature review in Section 3.2, on the information and 5G specifications contained in Chapter 2, and on the motivation in Section 1.1 which indicated that a simulated environment could effectively facilitate the study of 5G networks, the simulation environment illustrated in Figure 8 was designed. Its configuration comprises a simulated RAN, including a 5G UE and a gNB, alongside a 5GC and a network resource, exemplified by internet access. This setup accurately mimics the interactions of a UE within a real-world 5G cellular network.

Figure 8 – Experimental setup overview



Source: Created by the author (2025).

The simulation environment may incorporate a traffic generator, that transmits packets through the virtual UE network interface. This generator operates in three distinct modes: playback of stored videos, streaming of a live video feed, and transmission of UDP packets. In the first mode, stored videos are played to simulate the behavior characteristic of eMBB traffic. The live video feed mode streams a broadcast that represents URLLC

traffic. The third mode facilitates the transmission of UDP packets emulating mMTC traffic associated with the IoT. As detailed in (106), sparse UDP traffic effectively simulates IoT traffic corresponding with industrial automation use cases. This approach is simpler to implement than other widely used protocols, such as MQTT (107), and aligns with the mMTC axis definition presented in Subsection *2.1.3.1* by transmitting small volumes of non-delay-sensitive data. With the exception of the third mode, which also requires the creation of a server on the 5GC, all actions are performed on the UE. An illustrative implementation of the traffic generator is provided in Subsection **4.5.3**.

A packet capture experiment based on the environment described and the techniques from the reviewed work in Section 3.2 is also performed. The objective of this experiment is to collect network traffic packets, specifically targeting different application flows for subsequent analysis and potential reuse. To minimize the impact on NFs and ensure the collection of high-fidelity network data, passive packet capture techniques are recommended (108). This approach is essential to ensure that the capture process does not interfere with the observed traffic. Drawing from prior experience with traffic captures and the established file structure that facilitates the recording of detailed data and metadata, as well as insights from the experiment conducted in (59), it is proposed that the captured data should be stored in the PCAP file format. The implementation of this experiment is comprehensively described in Subsection **5.1.1**.

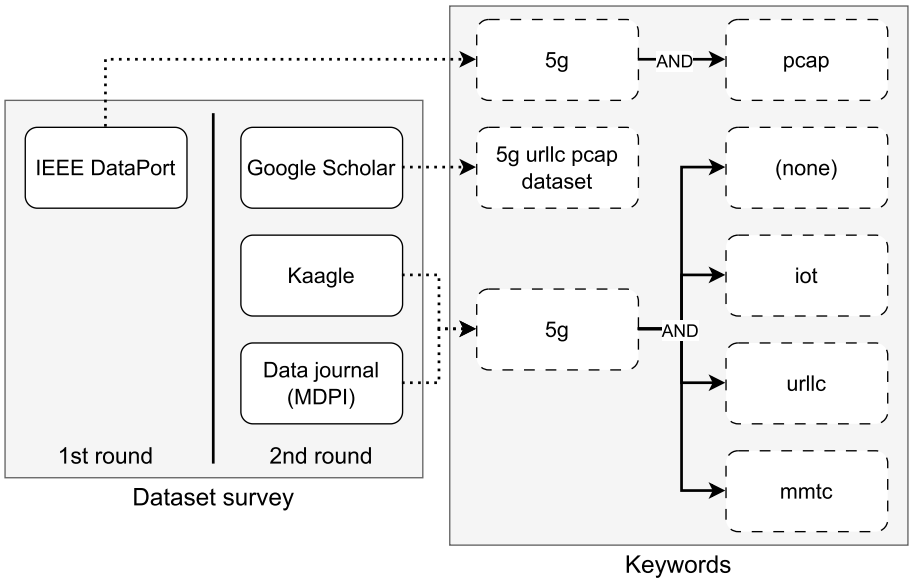
4.3 5G NETWORKS TRAFFIC DATASETS

Based on the findings presented in Section 3.3 and detailed in Table 4, no public datasets specifically targeting UE classification were identified. Consequently, a survey was conducted to identify 5G network traffic datasets, with particular emphasis on the PCAP file format due to its capacity to provide detailed information essential for the feature extraction phase in ML model training. In contrast, preprocessed formats such as Comma Separated Values (CSV) often limit the granularity of data. Additionally, it was imperative that the datasets encompass all three classes of 5G traffic: eMBB, URLLC, and mMTC. The data should originate from a real network environment rather than a fully simulated context, ensuring the representation of authentic network behavior and avoiding reliance on synthetic traffic generators.

With these criteria established, the survey was conducted in January 2025 to locate suitable 5G PCAP traffic datasets on IEEE DataPort (109). An overview of the survey process, including the consulted databases and keywords used, is depicted in Figure 9. The dataset titled “5G Traffic Datasets” (110) was initially selected; however, it was subsequently deemed insufficient due to the absence of mMTC traffic, as shown in Table 5 presented in the following paragraph. Following this, as represented in Figure 9, additional searches were performed across Google Scholar (111), Kaagle (112), and MDPI’s Data

journal (113) but these efforts did not yield any datasets that met the specified criteria. Ultimately, the "5G Campus Networks: Measurement Traces" (114) dataset emerged as the closest match. The search utilized keywords such as `5g urllc pcap dataset` on Google Scholar, along with combinations of `5g` and individual terms including `pcap`, `iot`, `urllc`, and `mmtc` to explicitly target datasets containing mMTC traffic.

Figure 9 – Dataset sources and keywords



Source: Created by the author (2025).

The dataset titled "5G Traffic Datasets" (110) comprises packet captures from a Commercial off-the-shelf (COTS) UE — Samsung Galaxy A90 5G — operating on the network of a major MNO in South Korea. The captures were conducted using the PCAPdroid application while the device interacted with various services, as detailed in Table 5. According to the authors, these interactions were performed sequentially and without background traffic to isolate the unique characteristics of each traffic type. The fourth column of Table 5 presents a classification based on the ITU recommendations (16) and definitions outlined in Subsection *2.1.3.1*, while the other columns are directly sourced from the dataset description.

The dataset "5G Campus Networks: Measurement Traces" (114) includes packet captures from an experimental 5G campus network, which features two Nokia ASiR antennas (one for 4G and one for 5G), a Ruckus ICX 7850 switch, and two Core Network (CN) instances (one Nokia NSA Core and one Open5GS SA Core). Additionally, two devices represented the UEs (a Nokia FastMile 5G Gateway and a WNC SKM-5xE). The 5G Stand-Alone (SA) traffic was generated using the MoonGen (115) open source software.

Based on their descriptions and configurations, the "5G Traffic Datasets" served as the eMBB and URLLC classes, while the "5G Campus Networks: Measurement Traces" served as the source for the mMTC class. The files selected from the first dataset were

Table 5 – Dataset capture description and service classification

Type	**Application**	**Main Protocol**	**Service Axis**
Live Streaming	YouTube Live	GQUIC	URLLC
	AfreecaTV	TCP	
	Naver NOW	TCP	
Stored Streaming	YouTube	QUIC	eMBB
	Netflix	TCP	
	Amazon Prime Video	TCP	
Video Conferencing	Zoom	UDP	URLLC
	MS Teams	UDP	
	Google Meet	UDP	
Metaverse	Zepeto	TCP	eMBB
	Roblox	RakNet	
Online Game	Teamfight Tactics	UDP	URLLC
	Battleground	UDP	
Game Streaming	GeForce Now	UDP	eMBB
	KT GameBox	UDP	

Source: Created by the author (2025). Adapted from (110).

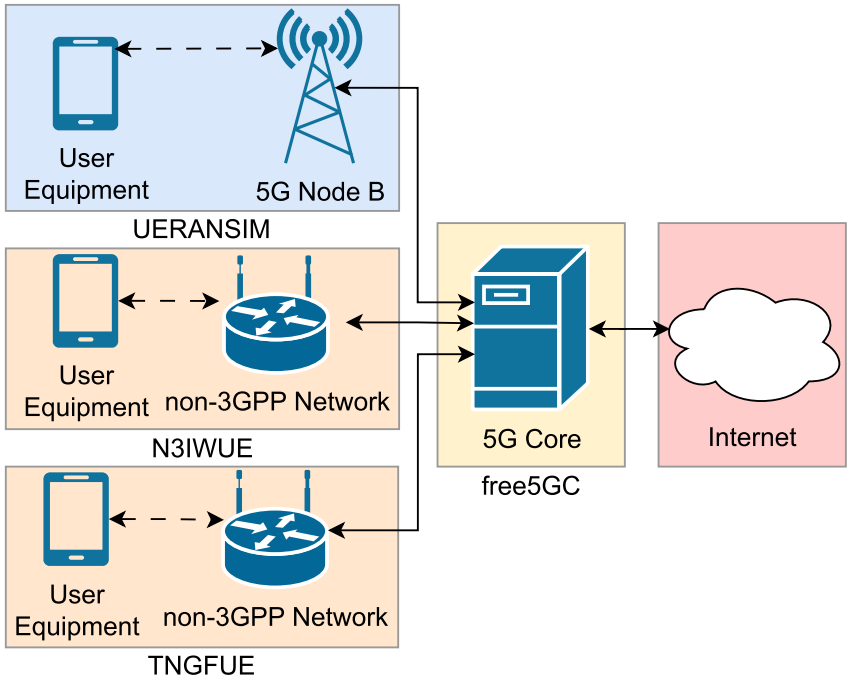
chosen according to our preliminary classification, the volume of available packets, and the feasibility of reproducing the experimental tasks as stipulated by the authors. This led to the selection of the `Youtube_cellular` and `naver5g3` files for the model training step. Conversely, the files from the second dataset were sourced from the 5G SA segment of the network, specifically located in the `SAUpload` and `UDPPing` folders. These data slices constituted the training dataset employed in the experiments analyzed in Subsection **5.2.3**.

4.4 FREE5GC AUTO DEPLOY

Given that GitHub is the most utilized code version control platform (116) and hosts free5GC's source code, an open-source tool search was conducted on February, 2024. As presented in (29) and (32), among the 301 results found, 23 were related to Kubernetes, and 27 to Docker (representing automated container deployment), while 35 used Shell Script (a programming language suitable for lightweight, task-oriented automation in Bash environments); however, it was determined that no similar open-source tool capable of deploying free5GC with UERANSIM (3GPP networks), N3IWUE, and TNGFUE (non-3GPP networks) in a pre configured form existed at that time.

In the light of the simulation environment outlined in Section 4.2 and the aforementioned tool survey, the free5GC Auto Deploy (FAD) tool was developed. FAD comprises a collection of scripts (available in Appendices A to E), that automate the installation of a 5G simulated environment. As illustrated in Figure 10, in addition to the outlined environment, FAD allows for the installation and configuration of non-3GPP NFs (highlighted in orange). It supports the advanced setup detailed in free5GC's official documentation (117), enabling deployment in the form of a software package stack. This configuration is particularly beneficial for projects that require experimentation, modularization, and the development of new NFs, while also offering enhanced flexibility for customization and network configuration. Notably, given its potential to enhance the reproducibility of research results, facilitate the study and future reuse of the source code, and the fact that this work received external public funding, FAD was released as FLOSS software in a GitHub repository (29).

Figure 10 – Outline of the environment deployed by FAD



Source: Created by the author (2025).

During the implementation process, several challenges were encountered, as reported in (31). The identified limitations and proposed solutions were formally documented and discussed in the free5GC official online repositories and forums (118). Code contributions focused on addressing minor issues, such as typos and execution flows, as well as automating network configuration tasks, such as firewall rule setup and UE deployment were made to the official repositories of free5GC (119) and TNGFUE (120), along with significant enhancements to clarify their supporting documentation (121), thereby making the solutions accessible to the community.

4.5 NWDAF FUNCTIONALITY IMPLEMENTATION

Based on the NWDAF definition and the 3GPP specifications summarized in Subsection *2.1.3.3*, a NWDAF functionality has been developed. The implementation consists of two primary Python scripts, which will be detailed in Subsection **4.5.2**. Additionally, four supporting scripts, described in Subsection **4.5.1**, were created to facilitate data processing steps necessary for preparing the data for the training stage of the pipeline outlined in Section 4.1. Subsection **4.5.3** presents the traffic generator scripts designed for integration into the environment to enable custom 5G traffic generation.

4.5.1 Data Preparation

The data preparation process involves several scripts, provided in Appendices F to L, designed to extract and structure the necessary information for ML models.

The `pcap_extract.sh` script — archived in Appendix F — extracts relevant data from the training and inference PCAP files and saves them in both JavaScript Object Notation (JSON) and CSV formats. Subsequently, the `dataset_CSV_characterization.py` script — in Appendix G — processes the CSV files generated by `pcap_extract.sh`, preparing five distinct subsets that include frame length, protocol label, a time series (comprising frame number and capture time), and source and destination IP addresses. These subsets are then utilized by the `stat-plotter.py` script — in Appendix H — for statistical analysis.

The `export_JSON.py` script — available in Appendix I — further enhances the data extraction process by preparing features from the JSON files created by `pcap_extract.sh` using the open-source `json2csv` module accessible in (122) and Appendix J. This script was designed based on prior experience (31) in PCAP data extraction and insights from relevant literature (97, 101), ensuring an effective approach to feature preparation. It structures the data in a manner compatible with the `box-plotter.py` — in Appendix K — script, which generates box plots for visual analysis. The CSV files produced by `export_JSON.py` contain a significantly greater level of detail compared to those generated by the original extraction, allowing for the creation of additional features that will be available during the training stage of the pipeline detailed in Section 4.1. As discussed in Section 3.3, the preparation process is configured to extract the 33 features listed in Table 3.

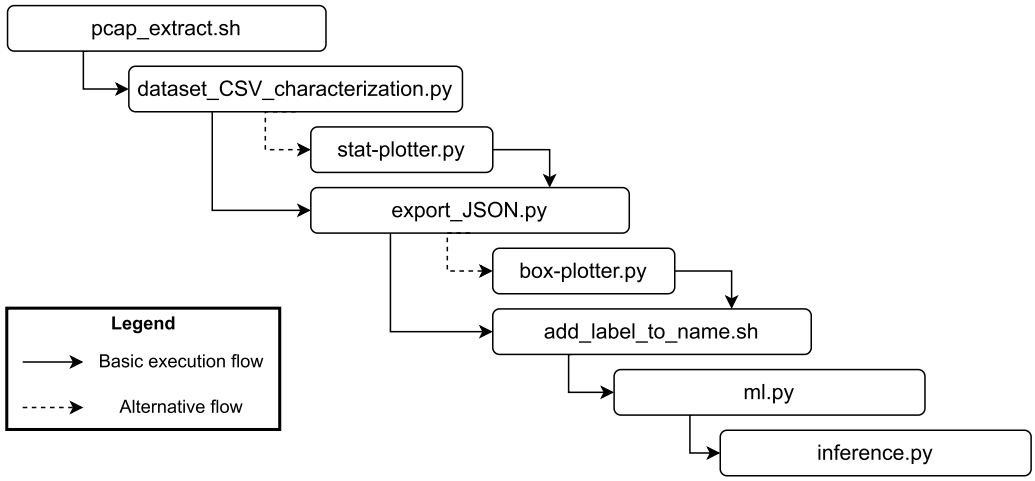
Additionally, the `add_label_to_name.sh` script — available in Appendix L — implements a manual labeling process that prepares the files for the automated steps of the training stage. Two visualization scripts, `stat-plotter.py` and `box-plotter.py`, are also included. The former generates graphs depicting statistics related to protocol labels, packet lengths, and the time series of captured packets, while the latter creates box plots of capture attributes, facilitating visual comparison and analysis.

4.5.2 Implementation Details and Execution

As the primary programming language used for implementing the proposal is Python, the scikit-learn (73) libraries were selected to ensure seamless integration and efficient computation of performance metrics. In addition to the accuracy metric specified by the 3GPP's Release 18 (61), the implementation also evaluates precision, recall, F1-score, and ROC AUC score, as defined in Subsection **2.2.2** and in Appendix Y, respectively.

The primary functionality based on the NWDAF specifications presented in Subsection *2.1.3.3* has been implemented in the `ml.py` and `inference.py` scripts. These scripts — archived in Appendices M and N — incorporate, respectively, the MTLF and AnLF logical functions detailed in Subsection *2.1.3.3*. From the pipeline perspective, `ml.py` and `inference.py` scripts implement the training and inference stages, as discussed in Section 4.1. Figure 11 depicts the execution order of the aforementioned scripts. The solid arrows indicate the relationships between the scripts, illustrating how the output of one script serves as the input for the subsequent script, while the dashed lines denote optional steps that produce the statistical graphs presented in Subsection **5.1.2**.

Figure 11 – Implemented functionality workflow



Source: Created by the author (2025).

Drawing on prior experience (30, 31), features with known low variability — due to their close association with stream characteristics, such as Hypertext Transport Protocol Secure (HTTPS) servers operating on port 443 by default — were excluded from the `ml.py` preprocessing step, as detailed in Table 6.

Table 6 – Features removed prior to model training

Packet_no	Source_IP	Destination_IP	Frame_protocols
IP_protocols	Source_port	Destination_port	

Source: Created by the author (2025).

Table 7 – Features utilized in model training

Timestamp	TCP_compl_data	IP_flag_reserved_bit
Time_delta	TCP_compl_ack	IP_flag_dont_fragment
Frame_type	TCP_compl_syn_ack	IP_flag_more_fragments
Frame_total_length	TCP_compl_str	Time To Live (TTL)
Frame_header_length	TCP_flags_bin	TCP_header_length
Frame_payload_length	TCP_window_size	Data_length
TCP_completeness	TCP_compl_syn	QUIC_packet_length
TCP_compl_reset	TCP_flags_str	QUIC_packet
TCP_compl_fin	TCP_window_size_scale	

Source: Created by the author (2025).

Furthermore, based on the literature review results presented in Section 3.2, the impact of including not only features such as packet timestamp and its total length but also window lengths and several flags on model performance was assessed in the evaluation analyzed in Chapter 5. Consequently, the 27 features listed in Table 7 were employed in the model training implemented in the `ml.py` script, which also includes a data balancing function that utilizes SMOTE (141) to balance the training data.

Finally, it is important to note that the current implementation of the NWDAF functionality is not fully integrated into the SBA as a NF. Therefore, the packet capture to create the PCAP files must be performed prior to inputting the data into the pipeline, and the classification results were not utilized by any other NFs. The source code for the implemented functionality has been made available on GitHub (41) under a copyleft license, allowing for study and reuse. Instructions for reproducing the implementation with alternative datasets are also provided.

4.5.3 Traffic Generator

As part of the implementation detailed in Subsection **4.5.2** and the traffic generator proposed in Section 4.2, a simple traffic generator has been implemented. This generator consists of three scripts available in Appendix O: `play-video.sh`, `udp-server.sh`, and `udp-client.sh`.

The `play-video.sh` script is capable of playing a playlist of stored YouTube videos or streaming a live broadcast from a Korean news channel on Naver TV (formerly Naver NOW), depending on the selected mode. The `udp-client.sh` script should be executed on the UE, while the `udp-server.sh` can be executed on the 5GC or on another server on the internet. The two UDP scripts facilitate the transmission of UDP packets in two configurations: at a fixed rate or within a probability-based interval range. In the fixed rate, it is possible to send packets in 100 Packets per Second (PPS) or a custom rate, as suggested in (106). The interval range configuration is implemented based on the behavior

of IoT devices, as described in (123). Additional details, including the source code and usage instructions for the traffic generator, are available in the GitHub repository (41).

Due to the necessity, as highlighted in Subsection **4.5.2**, of generating the PCAP files prior to pipeline execution, the traffic generator was run in an instance of the simulation environment described in Section 4.2. The packet capture experiments, conducted using the generator, to create the inference dataset presented in Subsection **5.1.1** were based on the dataset descriptions provided in Section 4.3. Notably, the generator simulates network behavior and enables not only creating PCAP datasets, but also using the traffic in tasks such as network performance tests.

4.6 SUMMARY

This chapter presented the methodology containing a ML pipeline and a 5G simulation environment, forming the basis for the experimental results in Chapter 5. The dataset survey conducted to help selecting the datasets used in the model training was outlined. The two datasets “5G Traffic Datasets” and “5G Campus Networks: Measurement Traces” selected for the model training were presented. Finally, the implementation of the environment, NWDAF-based functionality, and traffic generator were also detailed.

5 RESULTS AND DISCUSSION

This chapter presents the results obtained from the research and its corresponding discussions. Initially, Section 5.1 provides an overview of the outcomes associated with the created network dataset. Subsequently, Section 5.2 introduces the ML model experimental results along with relevant analyses. Finally, Section 5.3 includes discussions focused on the evaluation of model performance.

5.1 NETWORK DATASET

Based on the analyses that underscore the necessity of a PCAP dataset for UE classification, as presented in Section 3.3, the pipeline requirements for training and inference datasets outlined in Section 4.1, and the findings of the dataset survey presented in Section 4.3, a network packet capture dataset has been created, as detailed in Subsection **5.1.1**. Furthermore, Subsection **5.1.2** contains the frequency analyses of two characteristics — Protocol Label and Frame Length — of both training and inference datasets. Additionally, Section **5.1.3** presents the results of statistical hypothesis tests executed to compare the training and inference datasets. These datasets correspond to the dataset introduced in Section 4.3 and the newly created PCAP dataset detailed in Subsection **5.1.1**, respectively.

5.1.1 Packet Capture Dataset

The results in (59) indicate that using only GTP-U packets for multi-class classification can reduce model performance by up to 36.7% compared to using actual IP packets. In their work, the MedBIoT dataset (96), which contains captures of both Wi-Fi and Ethernet traffic, was replayed in a simulated 5G network to generate new network traffic, including the IP traffic captured in the UPF.

Consequently, by drawing on prior experience (30, 31), the convenience of having a virtual network interface readily available for packet capture, and the capturing methodology outlined in (59), the capture was conducted within the UPF. This UPF instance operated within a 5GC instance deployed in a simulated setup using the FAD tool, as described in Section 4.4. The simulated environment adhered to the configuration suggested in Section 4.2, utilizing UERANSIM as the UE and RAN simulator, and free5GC for the 5GC, with internet access to execute the traffic generator presented in Subsection **4.5.3** for the packet capture experiment.

The packet capture was performed by executing Wireshark's command-line utility, tshark (124), on the UPF's interface while the virtual UE executed tasks that replicated the experimental conditions described by the creators of the training datasets (110) and (114) regarding the data selected on the dataset processing stage for model trai-

ning. This process resulted in a PCAP dataset of simulated 5G traffic comprised by four PCAP files: `youtube-1M-1080p.pcap`, `naver-tv-1M.pcap`, `udp-100pps.pcap`, and `udp-nc-traffic-1k.pcap`.

The files `youtube-1M-1080p.pcap` and `naver-tv-1M.pcap` were created using the `play-video.sh` script from the traffic generator, representing the eMBB and URLLC service classes, respectively, with each containing approximately 1 million packets. The files `udp-100pps.pcap` and `udp-nc-traffic-1k.pcap` were created using the `udp-server.sh` and `udp-client.sh` scripts from the traffic generator presented in Subsection **4.5.3**. The server operated on the same Virtual Machine (VM) as free5GC, with the first file containing approximately 1 million packets generated by simulating an UDP burst traffic of 100 PPS — as described in (106) — and the second containing approximately 1 thousand UDP packets generated by a probabilistic interval range — as detailed in (123).

The dataset described above, used in the experimental inference phase, along with the data slices from the datasets (110) and (114) utilized to construct the training dataset, are publicly accessible on Zenodo (40). Additional details and reproduction instructions for the traffic generation experiment are available in the GitHub repository (41).

5.1.2 Frequency Analysis

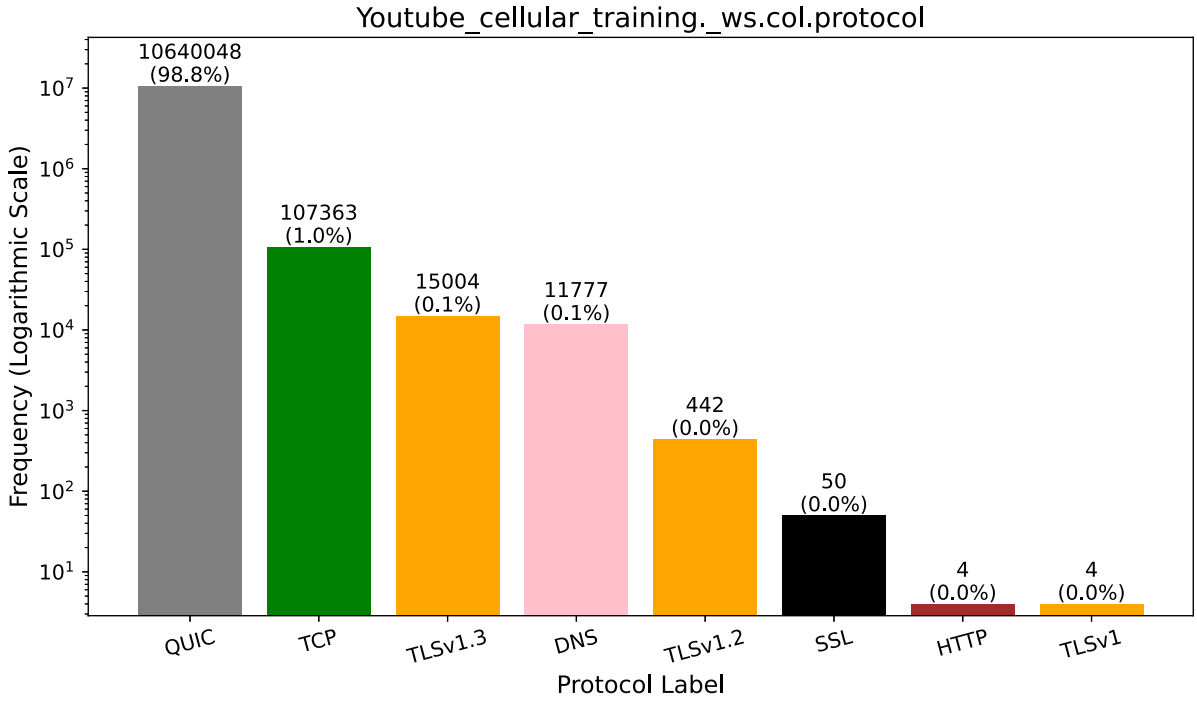
To improve the understanding of the created inference dataset, facilitate comparisons with the training dataset, and establish a foundation for the discussion in Section 5.3, a univariate analysis was conducted as part of the Exploratory Data Analysis (EDA) process, as recommended in (75). This analysis focused on two features from the PCAP files: Protocol Label (Section *5.1.2.1*) and Frame Length (Section *5.1.2.2*), which were selected due to their significant capacity of describing network streams.

5.1.2.1 Protocol Label

One analyzed feature concerns the protocol labels (`_ws.col.protocol`) assigned to each captured packet, which reflect the highest layer protocol detected by Wireshark. Their frequency of occurrence is illustrated in Figures 12–17.

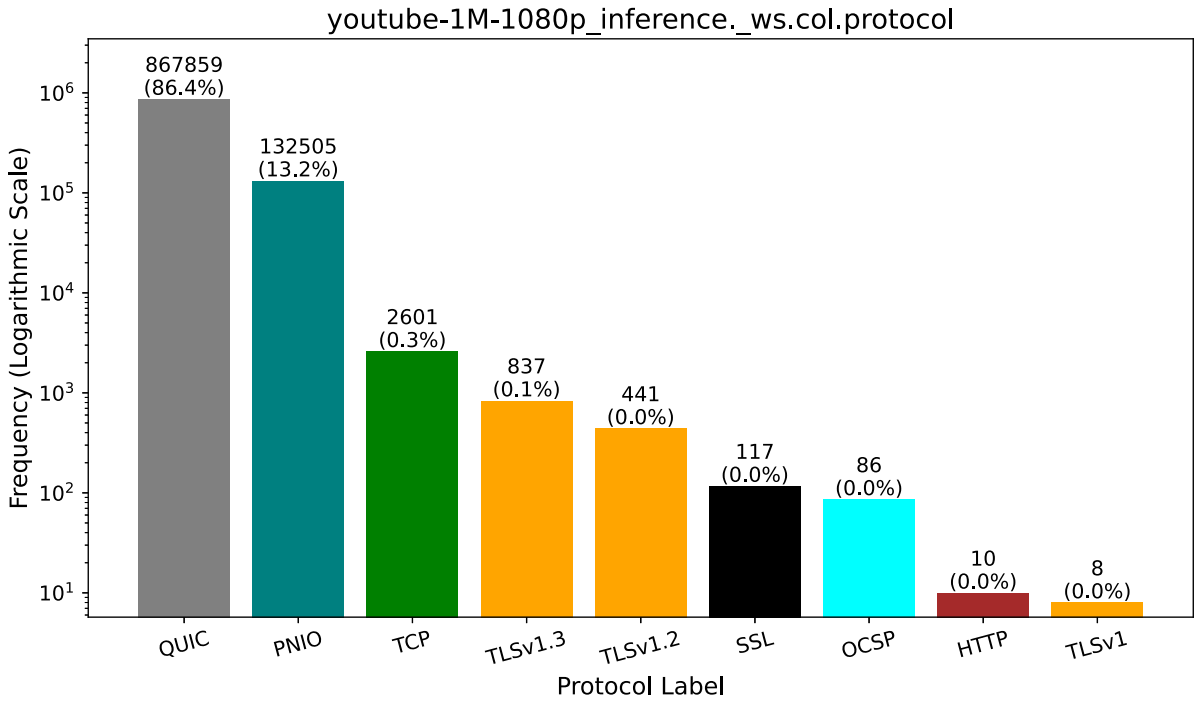
In the eMBB traffic, depicted in Figures 12 and 13, the QUIC protocol predominates, comprising 98.8% of the training dataset and 86.4% of the inference dataset. This prevalence is anticipated, as QUIC is the primary video transmission protocol utilized by YouTube, the platform generating the traffic. The video content itself constitutes the majority of the traffic due to its larger size compared to other web elements on a video-sharing platform. Other protocols, such as TCP, TLSv1.3, TLSv1.2, Secure Sockets Layer (SSL), Hypertext Transport Protocol (HTTP), and TLSv1, exhibit similar occurrence rates in both datasets. Notably, the training dataset includes 0.1% of Domain Name System (DNS) packets, likely due to the methodology employed by its authors in (110), which involved capturing packets

Figure 12 – Protocol label occurrence frequency in eMBB training dataset



Source: Created by the author (2025).

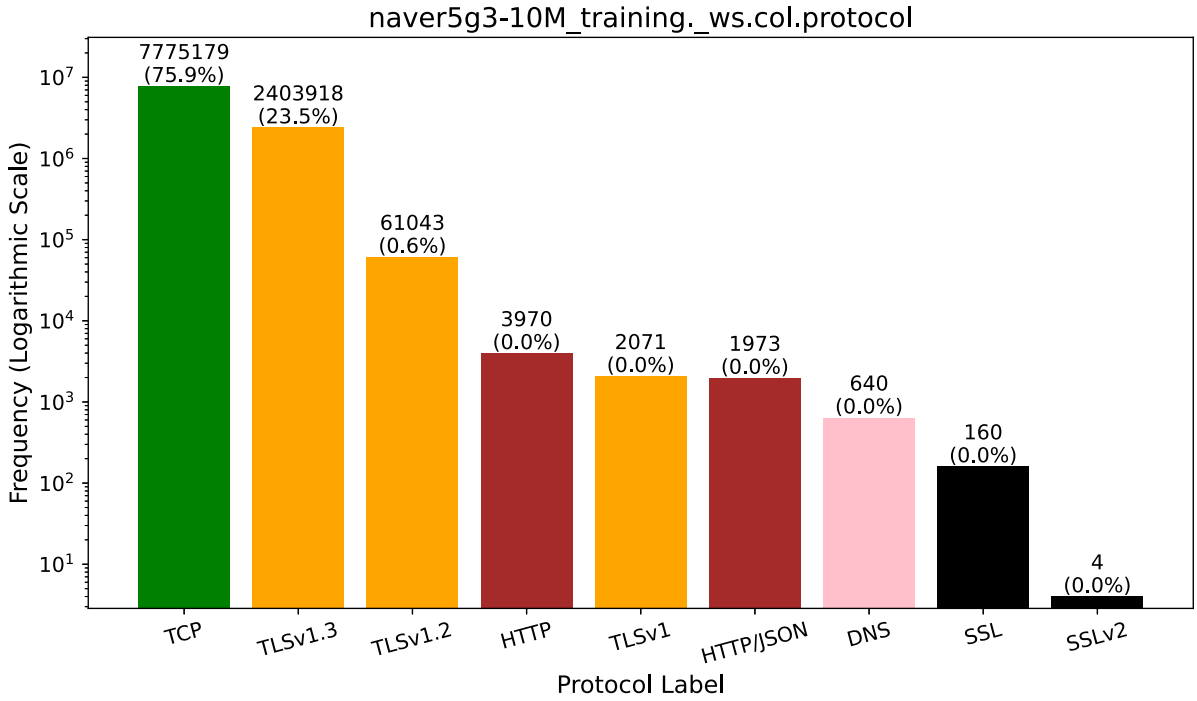
Figure 13 – Protocol label occurrence frequency in eMBB inference dataset



Source: Created by the author (2025).

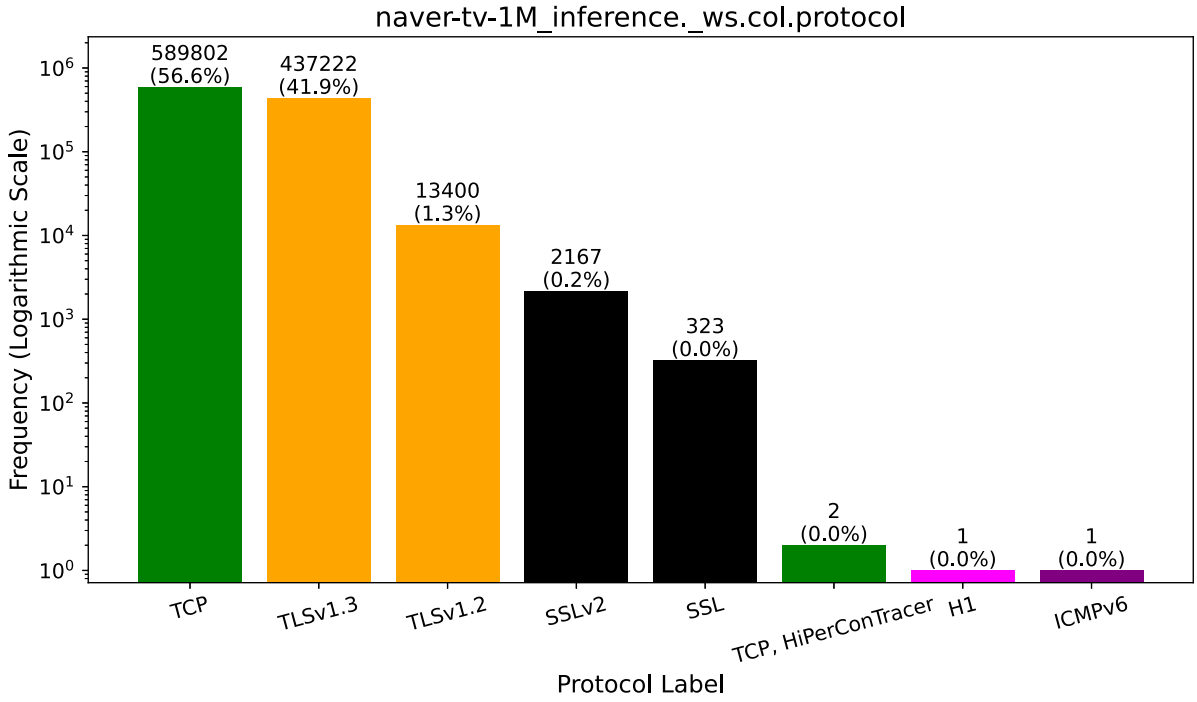
from the UE using PCAPdroid. This may have facilitated the capture of DNS queries that were mostly aimed at resolving Google domains, consistent with the behavior of an Android Operating System (OS) device streaming YouTube videos. The absence of this

Figure 14 – Protocol label occurrence frequency in URLLC training dataset



Source: Created by the author (2025).

Figure 15 – Protocol label occurrence frequency in URLLC inference dataset



Source: Created by the author (2025).

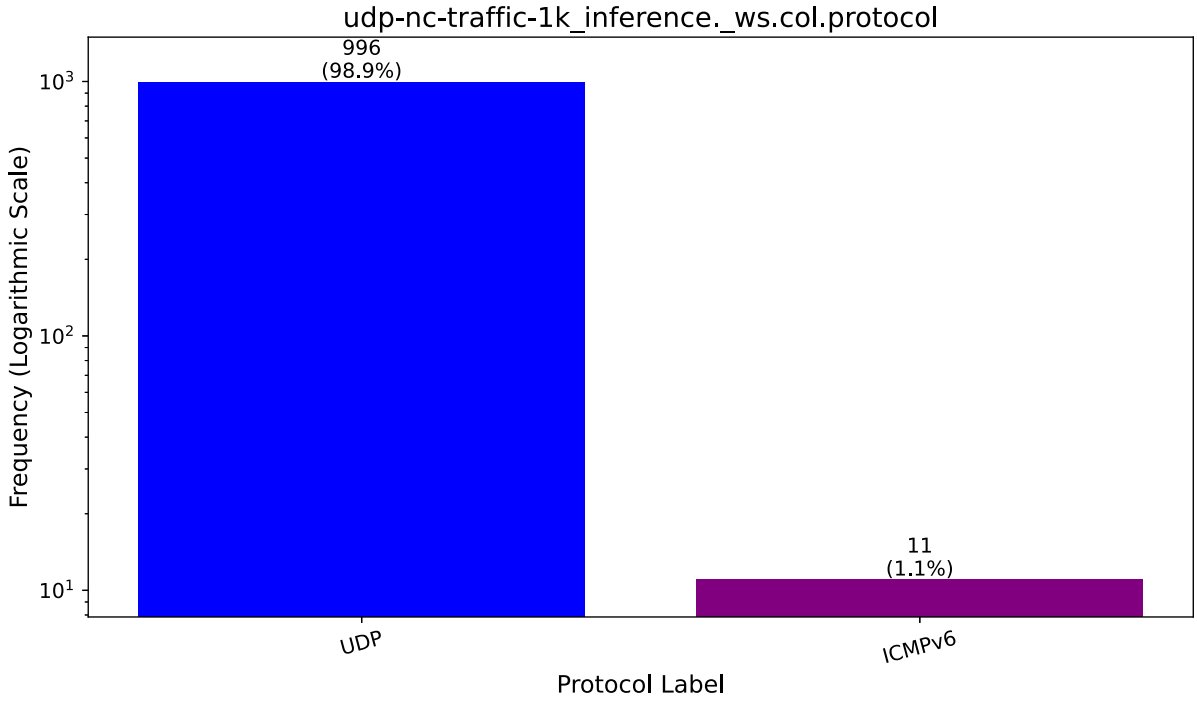
traffic in the inference dataset may be attributed to differences in the capture environment, as the inference capture was conducted in the 5GC with the UE being simulated in a Ubuntu Linux OS VM. Consequently, this traffic may have been encapsulated by other

protocols (e.g., DNS over TLS (DoT) or DNS over HTTPS (DoH)), leaked through the VM's other network interface, or not captured due to the relative limited capture duration of approximately 32 minutes, which was sufficient to generate 1 million packets while the video was played and the queries were cached. A combination of these factors is also plausible. In the inference dataset, notable labels include Online Certificate Status Protocol (OCSP) and PROFINET IO (PNIO). OCSP is utilized to verify the revocation status of X.509 HTTPS-related certificates and is enabled by default in Mozilla Firefox, the browser used for the eMBB experiment. PNIO facilitates data exchange between Ethernet-based field devices in IP-based protocols designed for Programmable Logic Controller (PLC) communication (125). The packets associated with this protocol were likely encrypted and used for communication with a Google-owned IP address, suggesting potential control synchronization, fingerprinting, or reporting activities.

As illustrated in Figures 14 and 15, the predominant protocol labels in the URLLC traffic for both training and inference datasets were TCP (75.9%/56.6%), Transport Layer Security (TLS)v1.3 (23.5%/41.9%), and TLSv1.2 (0.6%/1.3%). This distribution may be attributed to the operational characteristics of the Naver TV live stream player, which transmits its live video content via a combination of TCP and TLSv1.3. The absolute number of SSL packets increased in the inference dataset compared to the training dataset (323 vs. 160 packets), although it still accounted for less than 0.1% of the total share. Conversely, SSLv2 was captured only 4 times in the training dataset but reached 2167 packets (0.2%) in the inference dataset. Comparing the counts of SSL and SSLv2 with those of HTTP and HTTP/JSON in the training dataset suggests that some clear-text HTTP traffic may have been encrypted using SSL/SSLv2, as one method of securing HTTP payloads is through SSL. The higher proportions of TLSv1.3 and TLSv1.2 may also relate to HTTP protection. These patterns may have arisen from the capture environment (in UE vs. in the 5GC), as capturing unencrypted traffic is generally more feasible on the transmitting device. Similar to the eMBB experiment, the absence of DNS traffic in the inference dataset is believed to be due to analogous factors. Lastly, the inference dataset highlights three additional protocols: HiPerConTracer, H1, and ICMPv6. HiPerConTracer is associated with a tool that analyzes and diagnoses network connectivity issues with high precision and efficiency (126). H1 represents the SINEC H1 protocol, which is related to PLC communication for controlling Siemens industrial devices (127). The three packets corresponding to these protocols originated from a South Korean IP, likely linked to the Naver TV service. The ICMPv6 packet was a router solicitation unicast transmission sent to all routers within the Local Area Network (LAN), likely generated automatically by network devices, as the environment included a switch connecting the host to the internet.

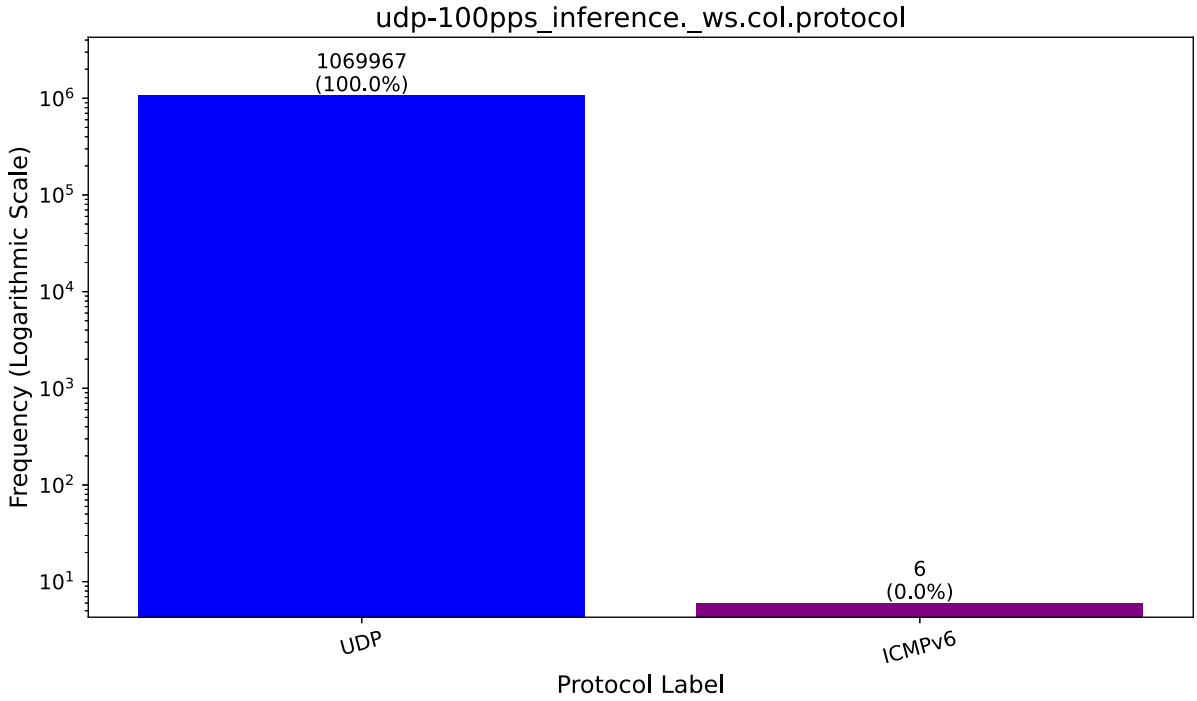
According to the description in (114) and the corresponding PCAPs, the predominant packet protocol label in the mMTC traffic was UDP, which accounted for 100% of the share in the training dataset. In the inference dataset, as illustrated in Figures 16 and 17,

Figure 16 – Protocol label occurrence frequency in mMTC inference dataset (probabilistic)



Source: Created by the author (2025).

Figure 17 – Protocol label occurrence frequency in mMTC inference dataset (burst)



Source: Created by the author (2025).

besides the prevalence of UDP packets, a small number (11 packets in the probabilistic capture and 6 packets in the 100 PPS capture) of ICMPv6 unicast packets were observed, likely generated automatically by the network switch present in the capture environment.

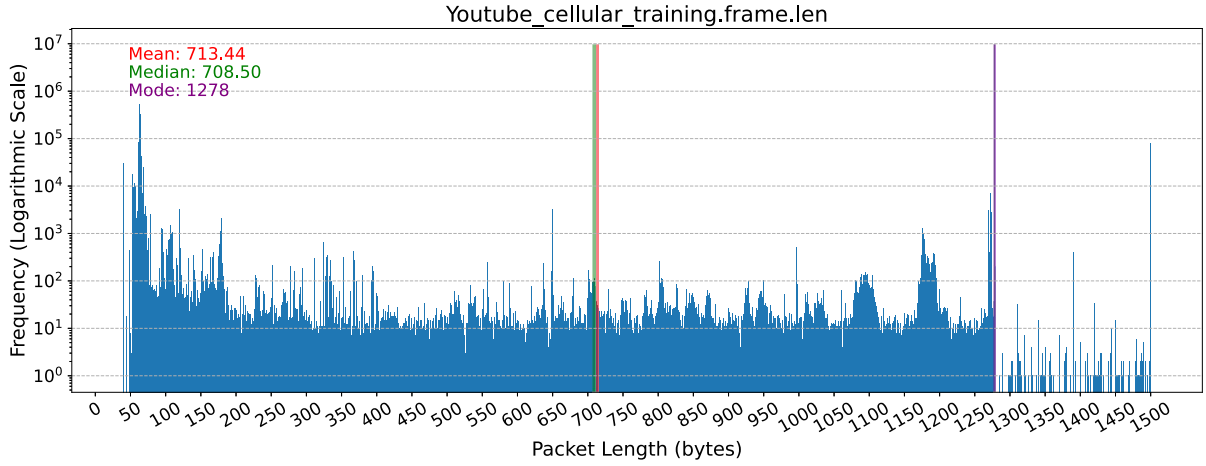
The observed differences in label occurrence between the training and inference datasets can be attributed to the distinct environments in which they were generated. The training dataset, as detailed in Section 4.3, comprised captures from physical network interfaces, whereas the inference dataset, outlined in Subsection **5.1.1**, collected packets from the UPF virtual network interface within a 5GC deployed in a simulated environment.

5.1.2.2 Frame Length

The second feature analyzed was packet length (`frame.len`). As indicated by the varying protocol types identified in Subsection *5.1.2.1* and given that the PCAP files contain various length measurements, including payload length, header length, and total packet length, the total packet length was selected for analysis to facilitate comparisons among the different captures. The results of this analysis are presented in Figures 18–21.

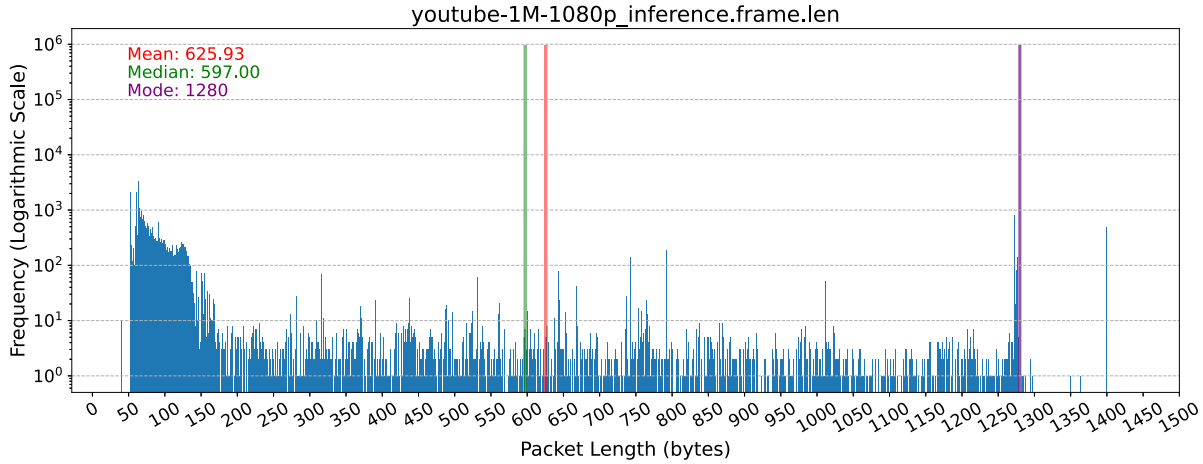
In the eMBB traffic, the majority of captured packets measured 1278 and 1280 bytes, accounting for approximately 87% of the training dataset and 96% of the inference dataset. This observation is further corroborated by the mode, which is highlighted in purple in both Figures 18 and 19. This pattern can be attributed to the video content being transmitted via YouTube, which aligns with the predominance of the QUIC protocol in both datasets (as discussed in Subsection *5.1.2.1*). Only a small fraction of packets exceeded 1280 bytes (0.8% in the training dataset and 0.1% in the inference dataset), and none of these utilized QUIC. This limitation may stem from the nature of QUIC, which cannot be fragmented (128) and thus adheres to a conservative Maximum Transfer Unit (MTU) value, as suggested in (129). In the inference dataset, the global MTU was 1400 bytes due to constraints on the simulated 5G NR interface, explaining the absence of larger packets in Figure 19, which contrasts with the training dataset. Additionally, the overall distribution of packet lengths differs between the two sets. As illustrated in

Figure 18 – Frame length distribution in eMBB training dataset



Source: Created by the author (2025).

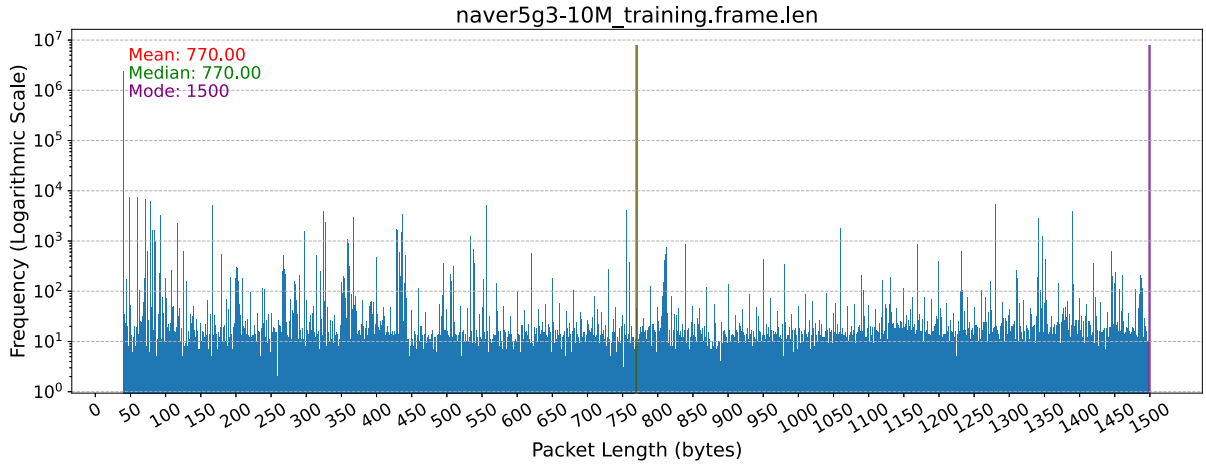
Figure 19 – Frame length distribution in eMBB inference dataset



Source: Created by the author (2025).

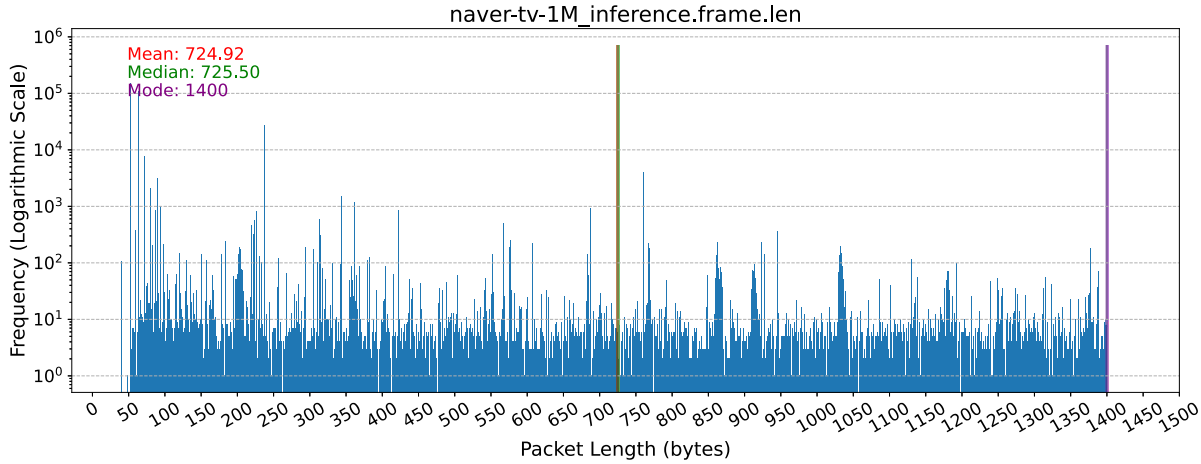
Figure 18, the training dataset exhibits five distinct frequency peaks for packet lengths exceeding 1,000 packets: (i) between 40 and 180 bytes; (ii) at 650 bytes; (iii) between 1174 and 1192 bytes; (iv) between 1270 and 1278 bytes; and (v) at 1500 bytes. Conversely, the inference dataset, depicted in Figure 19, shows only three peaks with frequencies exceeding 100 packets: (i) between 52 and 126 bytes; (ii) between 1272 and 1280 bytes; and (iii) at 1400 bytes. The frequency distribution for packet lengths outside these ranges exhibits a comparable pattern. Consequently, these observed distributions likely influenced the behavior of the mean and median values (respectively highlighted in red and green), as visually confirmed by Figures 18 and 19. Notably, the distance between these values experienced a slight variation when comparing the training and inference datasets, changing from 713.44 and 708.50 in the training dataset to 625.93 and 597.00 in the inference dataset, probably due to how the frequency peaks varied across the datasets.

Figure 20 – Frame length distribution in URLLC training dataset



Source: Created by the author (2025).

Figure 21 – Frame length distribution in URLLC inference dataset



Source: Created by the author (2025).

In the URLLC traffic, the majority (and consequently the mode) of captured packets measured 1500 bytes (approximately 75%) and 1400 bytes (approximately 68%), corresponding to the respective MTUs for the training and inference datasets. This behavior aligns with the characteristics of a video live stream application, as previously noted in the eMBB traffic analysis. The use of TCP and TLS for video transmission can also be inferred from the analysis in Subsection *5.1.2.1*. Figure 20 highlights another significant point in the training dataset at 40 bytes, representing approximately 24% of the dataset. In the inference dataset (represented in Figure 21), other notable lengths include 52 bytes (approximately 14%), 64 bytes (approximately 10%), and 237 bytes (approximately 3%). These packets may correspond to data transmitted alongside or prior to the actual video stream, such as live chat, recommended videos, and other video elements (e.g., descriptions, comments). The mean and median (colored in red and green in Figures 20 and 21) were consistent within the same dataset and their variation comparing the training and inference datasets occurred mostly because of the variation on the MTU that is directly related to the mode and the amplitude of the distributions.

In the mMTC traffic, the training dataset was generated using fixed lengths of 128 or 160 bytes. This caused the mean, median and mode to converge to the same values of 128 or 160, depending on the analyzed capture. In the inference dataset, due to the implementation of the traffic generator (which sent a status report message within the payload), the 100 PPS traffic exhibited lengths of 75 (the mode) and 48 bytes (with only 6 packets out of 1 million). The remaining 1,000 packets generated using the probabilistic approach had lengths of 72 bytes (approximately 59%, representing the mode), 71 bytes (approximately 40%), and 48 bytes (approximately 1%). The behavior of the training dataset aligns with the authors' description of generating a synthetic traffic burst (114), while the variability in the inference dataset is attributed to the dynamic payload implemented by the traffic generator. The frequency distribution of the 100

PPS capture exhibited a convergent mean and median of 61.50, whereas the probabilistic capture had a mean of 63.67 and a median of 71.

5.1.3 Hypothesis Tests

Hypothesis tests serve as a statistical tool for comparing two datasets by assessing the evidence against a null hypothesis (H_0). This null hypothesis typically asserts that there is no significant difference between the two groups, or that the data follows the same distribution. In conjunction, an alternative hypothesis (H_1) is often formulated, positing that a difference exists, indicating that the data follows a different distribution (130).

In the context of this work, hypothesis tests are employed to compare datasets. If a difference is identified (as discussed in Subsection **5.1.2**), it is essential to determine whether the difference in central tendency (e.g., mean or median) is statistically significant.

As suggested by (131), Mann-Whitney U test, as defined in (132), is applicable for hypothesis testing the distributions of two samples. The assumptions of this test are:

1. The samples are independent and identically distributed (iid)
2. The samples can be ranked

Within this test, the hypotheses are defined as:

- H_0: The distributions of all samples are equal
- H_1: The distributions of one or more samples are not equal

Additionally, as suggested by (130) and (133), Pearson's chi-squared test, defined in (134), is also suitable for hypothesis testing. Its assumptions include:

1. The samples used in the calculation of the contingency table are independent
2. Each cell of the contingency table contains 25 or more observations.

The hypotheses for this test are defined as:

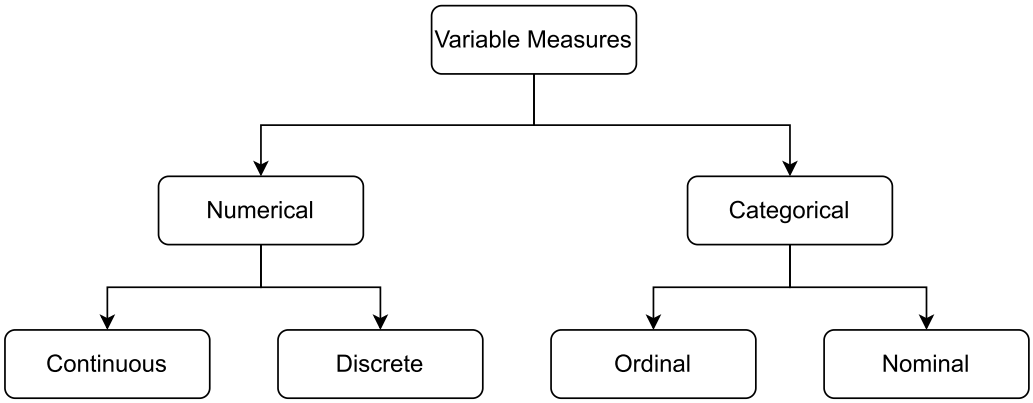
- H_0: There is no dependency between the samples (i.e., the samples are independent)
- H_1: There is a dependency between the samples.

Failing to reject the null hypothesis (H_0) in Pearson's chi-squared test indicates insufficient evidence to demonstrate a dependent relationship between the features. Consequently, a p-value exceeding the predetermined threshold suggests that any observed differences may be attributable to random chance.

As illustrated in Figure 22, variable measures are categorized as follows (135):

- Numerical continuous: An infinite number of values exist between two distinct points within a given range
- Numerical discrete: A finite amount of whole numbers are available within a specified range
- Categorical ordinal: Nominal data that can be ranked or sorted
- Categorical nominal: Nominal data that cannot be ranked or sorted

Figure 22 – Variable measures



Source: Created by the author (2025). Adapted from (136).

Table 8 illustrates the mapping of features utilized by the models (as detailed in Subsection **5.2.2**) based on these variable measures.

Table 8 – Feature measures

Measure	**Feature Name**
Numerical Continuous	Time_delta Timestamp
Numerical Discrete	QUIC_packet_length TCP_window_size Frame_total_length TCP_completeness TCP_header_length Frame_payload_length TCP_window_size_scale

Source: Created by the author (2025).

In this context, the Mann-Whitney U test was applied to the numerical continuous features, while Pearson's chi-squared test was employed for the numerical discrete features. These tests were selected due to the non-normal distribution and unequal variance observed in the training and inference datasets (as outlined in Subsection **5.1.2**), and because

applying the Mann-Whitney U test to the discrete features yielded inconclusive results given the sparse nature of the data.

Table 9 presents the p-values for each test executed comparing the training and inference datasets of each class. In the statistical tests performed, a p-value below the threshold (α) indicates that the null hypothesis (H_0) is rejected, whereas a p-value above this threshold signifies that the null hypothesis failed to be rejected. The raw test results detailed below are archived on Zenodo (39).

Table 9 – Hypothesis tests p-value for each scenario

Feature Name	**eMBB class**	**URLLC class**	**mMTC burst class**	**mMTC prob. class**
Time_delta	0	0	0	0
Timestamp	0	0	0.64554	0.3629
TCP_window_size	0	0	1	1
Frame_total_length	0	0	0	0
QUIC_packet_length	0	1	1	1
TCP_completeness	0	0	1	1
TCP_header_length	0	0	1	1
Frame_payload_length	0	0	0	0
TCP_window_size_scale	0	0	1	1

A common threshold value, corresponding to a 95% confidence level, is $\alpha = 0.05$, which is considered the minimum acceptable level for statistical tests involving numerical variable measures (131, 135). Therefore, as seen on Table 9, the distribution of features in the eMBB datasets was statistically significantly different in the training and inference datasets, as all tests returned p-values of 0. In the URLLC class, the `QUIC_packet_length` feature showed no significant difference with a p-value of 1, while all other features indicated significant differences, suggesting that the distribution differences between the training and inference datasets were still statistically relevant. In contrast, for the mMTC scenarios, out of the nine features tested, only three (`Time_delta`, `Frame_total_length`, and `Frame_payload_length`) had statistically significant differences on their distributions, while the remaining features did not. Based on these results, it can be concluded that the differences between the training and inference datasets used in the mMTC scenarios were not statistically significant, while the differences in the datasets utilized in eMBB and URLLC were.

5.2 MACHINE LEARNING MODELS FOR CLASSIFICATION

Considering the pipeline outlined in Section 4.1 and the 3GPP specifications detailed in Subsection *2.1.3.3*, which stipulate that ML models may be employed for data analytics tasks in 5G networks, this section presents the results of a ML performance evaluation

experiment focused on UE classification within the 5G context. The models tested in this section comprise: Linear Regression (LR), Histogram-based Gradient Boosting (HGB), Light Gradient Boosting Machine (LGBM), Multilayer Perceptron (MLP), Random Forest (RF), Linear Support Vector Classification (SVC), eXtreme Gradient Boosting (XGB), Decision Tree (DT), AdaBoost, Stacking and Voting. Subsection **5.2.1** describes the model tuning processes performed. Additionally, the model feature importance is analyzed in Subsection **5.2.2**. Then, the model performance results are detailed in Subsection **5.2.3**.

5.2.1 Model Tuning

Initially, as suggested in (137), a model hyperparameter optimization was attempted using the Differential Evolution (138) method. However, it could not converge with the parameters listed in the `model_tuning.py` script — available in Appendix P — of the `NWDAF_ml` module.

Given the higher level of explainability and ease of visualization of the DT model, as well as the customization possibilities of the Stacking and Voting ensemble models, manual parameter tuning was performed on these models.

The DT model was analyzed via the `dt_visualization.py` script — Appendix Q. After the initial analysis, the three parameters `min_samples_leaf` (which controls the minimum number of samples required at a tree leaf node), `min_samples_split` (the minimum number of samples required to enable splitting an internal node), and `max_depth` (the maximum depth of the tree branches) were adjusted to optimize the purity of the tree nodes, avoid having node splits with a very low number of instances, and balance the tree split decisions avoiding overfitting.

Considering that a Stacking model is suggested as a way to improve the model performance of base classifiers using another layer of classifiers (75), and that, as shown in (59) and (97), ensemble techniques may improve the classification performance, the ensemble based classifiers were also calibrated. By leveraging the customization possibilities of the Stacking and Voting ensemble models, the models with the best in class results were used as the base classifiers of the Stacking model, additionally, the Voting model had also the model vote weight calibrated aiming for avoid overfitting.

5.2.2 Feature Importance

To understand the weights assigned to the learned features by the models, average feature importance was extracted during model training and is presented in Tables 10 and 11. Among the eleven models tested, only the DT and RF models provided the `feature_importances_` function in Scikit-learn. Notably, as discussed in Subsection **5.2.1**, the DT model exhibits a higher level of explainability. After three executions, on average, both models consistently assigned the highest weight to `Time_delta`, with 51.12% for DT

and 20.89% for RF. This prominence is supported by the fact that, among the available features, `Time_delta` exhibited the highest variability compared to the features with statistically significant variations in their distribution, as seen in Subsection **5.1.3**.

Table 10 – Average feature importance for DT

Feature Name	**Weight**
Time_delta	0.5111874
TCP_window_size	0.4868625
Frame_total_length	0.0009336
Frame_payload_length	0.0005355
TCP_header_length	0.0000103
TCP_completeness	0.0000050

Source: Created by the author (2025).

Table 10 indicates that the next significant feature for DT was `TCP_window_size` with 48.69%. The remaining features — `Frame_total_length`, `Frame_payload_length`, `TCP_header_length`, and `TCP_completeness` — accounted for just approximately 0.15% of the weight, while 20 features were excluded by the model. This outcome is consistent with the available features outlined in Subsection **4.5.1** and the hyperparameter tuning discussed in Subsection **5.2.1**. As noted in Subsection **5.2.1**, the limited depth of the DT directly affected the weight distribution, as each decision or node split relies on a feature, and the number of splits correlates with the branch depth.

Table 11 – Average feature importance for RF

Feature Name	**Weight**
Time_delta	0.2088668
QUIC_packet_length	0.1620190
TCP_window_size	0.1387653
Frame_total_length	0.1321380
TCP_completeness	0.0986426
TCP_header_length	0.0903932
Frame_payload_length	0.0834209
TCP_window_size_scale	0.0808604
Timestamp	0.0048154

Source: Created by the author (2025).

As shown in Table 11, the RF model selected a greater number of features (9 compared to 6 in DT). The top three features included `Time_delta` (20.89%), `QUIC_packet_length` (16.20%) and `TCP_window_size` (13.88%), time and length-related packet features. Additionally, `Frame_total_length` (13.21%), `TCP_completeness` (9.86%), `TCP_header_length` (9.04%), `Frame_payload_length` (8.34%), and `TCP_window_size_scale` (8.09%) received a similar importance demonstrating the prevalence of length-related features.

Compared to the mentioned features, `Timestamp` (0.48%) received considerably less importance, probably due to the weight assigned to `Time_delta`, with the other 17 features discarded by the model. This more distributed weight allocation aligns with the RF model algorithm, which employs multiple DTs and feature bagging to mitigate bias and overfitting (139, 140).

5.2.3 Model Performance

The experimental step can be simplified by testing various ML algorithms through their libraries, with the evaluation of the most suitable options left to domain specialists or based on the results achieved during the experimentation.

Effective model performance evaluation is a crucial component of ML pipelines, as it serves as a benchmarking mechanism and helps in determining the reliability and generalization levels of developed models in real-world applications. The specification (61) only requires model accuracy as a performance metric, which is measured by comparing inference results against ground truth data. Following the pipeline of Section 4.1, accuracy and the extra four metrics described in Subsection **2.2.2** were used to evaluate the performance of the trained models in the training (Subsection *5.2.3.1*) and inference (Subsection *5.2.3.2*) stages. The raw model performance results detailed on the following sections are archived in Appendix S and on Zenodo (39).

5.2.3.1 Training

Out of a total of 22,023,650 samples in the training dataset outlined in Section 4.3, 70% (15,416,555 samples) were utilized for model training, while the remaining 30% (6,607,095 samples) were reserved for testing. The results of the F1-score metric of the testing are detailed in Table 12. The "average" metric refers to the average of all classes, with classes 0, 1 and 2 representing eMBB, URLLC, and mMTC, respectively.

Considering the average performance of three executions, all models achieved performance metrics exceeding 99%. These high performance results were anticipated due to the substantial volume of training data available. LR is commonly used as a baseline model, so it was included in the implementation. However, when compared to the other linear model (Linear SVC), it exhibited the lowest performance results. Concerning the average F1-score displayed on Table 12, the manually calibrated and ensemble models closely surpassed the linear models with F1-scores between 99.61% (Voting and DT) and approximately 99.8% (AdaBoost and Stacking). MLP and HGB exhibited 99.96% and 99.98% F1-score performance, while the remaining models (XGB, LGBM and RF) achieved more than 99.99%. Based on these results, along with those derived from the Inference presented in Section *5.2.3.2*, it is evident that the models manifested signs of overfitting in relation to the training data. The Appendix R contains the confusion matrices of the test

Table 12 – Training average F1-score performance results per class

Model	Class 0 F1-score	Class 1 F1-score	Class 2 F1-score	Average F1-score
RF	0.99999374	0.99999342	1.0	0.99999576
LGBM	0.99999337	0.99999287	0.99999985	0.99999541
XGB	0.99998299	0.99998203	0.99999993	0.99998842
HGB	0.99983398	0.99982529	0.99999985	0.99988738
MLP	0.99948346	0.99946733	0.99999035	0.99965001
Stacking	0.99707863	0.99694395	0.99999690	0.99802409
AdaBoost	0.99695374	0.99685454	0.99994667	0.99793582
DT	0.99424371	0.99402167	0.99999529	0.99612089
Voting	0.99421445	0.99403070	0.99995764	0.99610113
SVC	0.99419067	0.99399588	0.99996766	0.99608524
LR	0.99363267	0.99399833	0.99940700	0.99570696

Source: Created by the author (2025).

step and illustrate that the majority of FPs and FNs occur between eMBB and URLLC samples. This phenomenon may be attributed to the similarities in the characteristics of the network services generating these types of traffic, as described in Subsection **4.5.3**, and further discussed in Subsection **5.1.2**. Additionally, the Appendix S contains the tables with the all the results from the performance metrics detailed in Subsection **2.2.2**.

Another evaluation perspective is the performance of various models based on their training times, as presented in Table 13. The table includes three key measurements for each of the eleven models: training time, disk write time, and total training time. Training time refers to the duration required to train each model, while disk write time indicates the time spent to save the trained model to disk. The total time is the sum of these two measurements, providing insight into the overall efficiency of model deployment, particularly concerning in dynamic environments that necessitate frequent retraining.

Among the models analyzed in three runs, DT exhibited the best training time at approximately 23 seconds, and a disk write time of 0.28 Milliseconds (ms). LR followed with the second-best training time of around 31 seconds and a disk write time of 0.49 ms. LGBM ranked third, requiring approximately 40 seconds for training and 4.6 ms for disk writing. Notably, SVC demonstrated the highest efficiency in disk write time, yet it ranked sixth in training time. In contrast, DT, a more complex model than LR, exhibited superior training efficiency due to the parameter tuning described in Subsection **5.2.1**, which limited tree depth and resulted in a smaller model occupying only 3.0 Kilobytes (kB) on disk. Linear SVC and LR also had minimal disk space requirements, at 1.8 kB and 1.9 kB, respectively. Conversely, RF required over 9 seconds for disk writing, consuming 9.5 Megabytes (MB). The training time correlated with model complexity, with ensemble models being the slowest; RF's prolonged duration stemmed from insufficient hyperparameter tuning, based

Table 13 – Model average training time

Model	Average Training Time (ms)	Average Disk Write Time (ms)	Total Average Training Time (ms)
DT	22984.7205	0.2799	22985.0003
LR	31293.6348	0.4580	31294.0927
LGBM	39541.0260	4.5923	39545.6183
XGB	89135.0785	3.4888	89138.5673
HGB	115260.8451	3.7594	115264.6045
SVC	128639.4635	0.2562	128639.7197
MLP	431639.5728	0.5431	431640.1159
AdaBoost	487523.4267	1.0677	487524.4944
Voting	523247.1393	1.1789	523248.3183
RF	719960.1841	9.2561	719969.4402
Stacking	3366891.2883	1.4558	3366892.7441

Source: Created by the author (2025).

on the tree visualizations available in (41), which revealed considerably deeper trees. This contrast highlights the trade-offs between training efficiency and disk performance across different models.

5.2.3.2 Inference

To further validate the trained models, the inference step was executed to assess their performance in a real-world scenario using previously unseen data. The results of the inference process, including accuracy, weighted recall (as defined in Appendix X), and F1-score, are presented in Tables 14–17. Precision was consistently 100% across all inferences and is therefore omitted to conserve space. The number of samples used for inference was 1,004,464 for eMBB, 1,042,918 for URLLC, 1,069,973 for mMTC (burst), and 1,007 for mMTC (probabilistic) classes. The dataset used for this process was created as described in Subsection **5.1.1**.

In the eMBB class (results shown in Table 14), all models, as illustrated in the confusion matrices in Appendix T, achieved accuracy, recall, and F1-scores exceeding 99%. This performance aligns with expectations based on the results presented in Subsection *5.2.3.1*, which indicated overfitting to the training data.

As shown in Table 15, the URLLC class, the SVC, LR, and DT models achieved the highest F1-scores in the URLLC class, exceeding 99%. The Voting model scored approximately 98.87%, while the other models exhibited subpar performance due to the overfitting. Despite this overfitting, the linear models effectively identified URLLC packets, whereas DT and Voting models benefited from the tuning described in Subsection **5.2.1**. Given the experimental context outlined in Subsection **5.1.1** and the results in Subsec-

Table 14 – eMBB inference performance results

Model	**Accuracy**	**Recall**	**F1-score**
RF	0.99839616	0.99839616	0.99919744
LGBM	0.99829660	0.99829660	0.99914758
XGB	0.99821497	0.99821497	0.99910669
HGB	0.99817614	0.99817614	0.99908724
MLP	0.99770027	0.99770027	0.99884881
AdaBoost	0.99750613	0.99750613	0.99875151
Stacking	0.99653845	0.99653845	0.99826623
Voting	0.99598891	0.99598891	0.99799042
DT	0.99598194	0.99598194	0.99798692
SVC	0.99591822	0.99591822	0.99795494
LR	0.99591723	0.99591723	0.99795444

Source: Created by the author (2025).

Table 15 – URLLC inference performance results

Model	**Accuracy**	**Recall**	**F1-score**
SVC	0.99999041	0.99999041	0.99999521
LR	0.99524891	0.99524891	0.99761880
DT	0.99421335	0.99421335	0.99709828
Voting	0.97760035	0.97760035	0.98867332
Stacking	0.27436193	0.27436193	0.43058715
MLP	0.13418696	0.13418696	0.23662230
LGBM	0.13289731	0.13289731	0.23461493
HGB	0.13191449	0.13191449	0.23308208
XGB	0.11727768	0.11727768	0.20993470
RF	0.01483722	0.01483722	0.02924058
AdaBoost	0.00000096	0.00000096	0.00000192

Source: Created by the author (2025).

tion *5.2.3.1*, it can be inferred that SVC, LR, DT, and Voting demonstrated superior generalization of the features learned compared to the other models. The matrices in Appendix U confirm that the majority of misclassifications were associated with the eMBB class, which is attributed to the operational similarities in the services generating network traffic for theses classes, as discussed in Subsections *5.1.2.2* and **5.2.2**.

The results in Tables 16 and 17 indicate that even when the mMTC traffic similar to the training data (i.e., burst UDP traffic) was applied during inference, most models still failed to accurately classify packets. Notably, AdaBoost achieved an F1-score exceeding 99.99% for classifying mMTC burst packets. However, due to its subpar performance in the URLLC class (as shown in Table 15), it can be concluded that, compared to the other models, AdaBoost overfitted on the eMBB and mMTC data rather than on the eMBB and URLLC classes. As discussed in Subsection **5.1.3**, this limitation arises from

Table 16 – mMTC probabilistic inference performance results

Model	**Accuracy**	**Recall**	**F1-score**
AdaBoost	0.01390268	0.01390268	0.02742409
Voting	0.01290963	0.01290963	0.02549020
DT	0.01092354	0.01092354	0.02161100
HGB	0.01092354	0.01092354	0.02161100
LGBM	0.01092354	0.01092354	0.02161100
RF	0.01092354	0.01092354	0.02161100
XGB	0.01092354	0.01092354	0.02161100
MLP	0.01092354	0.01092354	0.02161100
Stacking	0.01092354	0.01092354	0.02161100
SVC	0.01092354	0.01092354	0.02161100
LR	0.00297915	0.00297915	0.00594059

Source: Created by the author (2025).

Table 17 – mMTC burst inference performance results

Model	**Accuracy**	**Recall**	**F1-score**
AdaBoost	0.99998878	0.99998878	0.99999439
LR	0.00003830	0.00003830	0.00007660
Voting	0.00002990	0.00002990	0.00005980
RF	0.00000561	0.00000561	0.00001120
XGB	0.00000561	0.00000561	0.00001120
DT	0.00000561	0.00000561	0.00001120
HGB	0.00000561	0.00000561	0.00001120
LGBM	0.00000561	0.00000561	0.00001120
SVC	0.00000561	0.00000561	0.00001120
MLP	0.00000561	0.00000561	0.00001120
Stacking	0.00000561	0.00000561	0.00001120

Source: Created by the author (2025).

significant differences between the mMTC data in the training and inference datasets. The confusion matrices in Appendices V and W reveal that, with the exception of the Linear SVC and Stacking models — both of which misclassified a substantial number of mMTC packets as URLLC — other models incorrectly classified mMTC packets as eMBB traffic in both burst and probabilistic scenarios. This misclassification persisted despite findings in Subsection *5.1.2.2*, which demonstrated that mMTC traffic substantially differentiated itself based on packet length variation, and Subsection **5.2.2**, which highlights the models' focus on the length-related features. Additionally, Appendix X complements the definitions in Subsection **2.2.2** and validates the results in this subsection by comparing not only different metrics of a given model but also the metrics of different models, explaining how they might converge to similar values.

Another aspect of the inference process is the time required for classification. Measuring this metric is crucial in environments with a high number of devices, where numerous classifications must be performed. The inference time was measured, with the average results from three runs displayed in Tables 18 and 19. These tables include classification time, which refers to the duration needed to classify the inference data, and total inference time, which encompasses the entire process of generating an inference response (loading the model from disk, preparing the data for classification — such as storing and then removing labels from the actual data — executing the classification, and calculating performance results).

Table 18 – eMBB, URLLC, and mMTC burst average 3 runs inference time

Model	**Average Classification Time (ms)**	**Model**	**Total Average Inference Time (ms)**
DT	29.1556	DT	382.5519
LR	43.7871	SVC	421.5565
SVC	45.9439	LR	443.8588
XGB	262.5967	MLP	844.9059
MLP	470.9609	XGB	1085.5019
LGBM	751.0224	LGBM	1212.5192
RF	1148.6976	RF	1528.0171
AdaBoost	1666.1416	AdaBoost	2020.9529
Stacking	1803.3771	Stacking	2180.9692
Voting	1850.9546	Voting	2208.9829
HGB	2003.1748	HGB	2376.4723

Source: Created by the author (2025).

Table 18 contains the inference time results for the classes with approximately 1 million packets. These classes were grouped due to their similar scale. On average, DT was the fastest model, achieving approximately 29 ms for classification and 383 ms for total inference time. It was followed by the linear models, LR and SVC, with classification times of approximately 44/444 ms and 46/422 ms, respectively. Other models required significantly more time for packet classification: XGB took approximately 263 ms for classification and 1,086 ms in total; MLP required about 471/845 ms; LGBM approximately 751/1,213 ms; RF approximately 1,148/1,528 ms; AdaBoost approximately 1,666/2,021 ms; Stacking approximately 1,803/2,181 ms; Voting approximately 1,850/2,209 ms; and HGB approximately 2,003/2,376 ms. These results align with the ones observed during model training (Section *5.2.3.1*), indicating that more complex models require more time for classification and related tasks. The custom ensemble models (Stacking and Voting), exhibited poorer time performance due to the combination of multiple models. DT outperformed the linear models (SVC and LR) due to the parameter tuning described in Subsection **5.2.1**, which limited tree size.

Table 19 – mMTC probabilistic average 3 runs inference time

Model	Average Classification Time (ms)	Model	Total Average Inference Time (ms)
DT	0.7357	DT	1.7479
LR	0.7566	SVC	1.8929
SVC	0.7627	LR	2.0565
MLP	1.0639	MLP	3.1155
LGBM	1.9073	LGBM	5.1222
HGB	4.5908	HGB	7.9354
XGB	5.2582	AdaBoost	8.8221
RF	5.8492	XGB	10.1210
AdaBoost	7.0638	Voting	10.4994
Stacking	8.5395	Stacking	10.5768
Voting	8.6442	RF	11.3975

Source: Created by the author (2025).

In the mMTC class with probabilistic traffic, Table 19 indicates that as the number of packets to be classified decreased, the performance ranking shifted, and the time differences among the models were significantly reduced. The fastest model in this scenario was DT, with approximately 0.74 ms for classification and 1.75 ms in total, closely followed by LR at approximately 0.76 ms for classification and 2.06 ms in total. Linear SVC ranked third with approximately 0.76 ms for classification and 1.89 ms in total. The other models performed as follows: MLP (approximately 1.06/3.12 ms), LGBM (approximately 1.90/5.12 ms), HGB (approximately 4.59/7.94 ms), XGB (approximately 5.26/10.12 ms), RF (approximately 5.85/11.40 ms), AdaBoost (approximately 7.06/8.82 ms), Stacking (approximately 8.54/10.58 ms), and Voting (approximately 8.64/10.50 ms). Despite the changes in ranking, the trend of DT being followed by the linear models, with ensemble models occupying the last positions, remained consistent.

Notably, total inference time was influenced by the implementation of statistical measurements, as detailed in Appendix N. In terms of measured times, DT consistently emerged as the fastest model for classification, outperforming the linear models (LR and SVC), which ranked among the top three classification models. XGB maintained a consistent fourth place in classification time during the inferences with a larger number of packets. In contrast, HGB exhibited the poorest classification and total inference times across all inferences with 1 million packets, while the ensemble models (AdaBoost, Stacking and Voting) ranked last due to the added complexity of combining multiple models.

5.2.3.3 Cross Validation

To deeply evaluate the results from the test phase displayed in Subsection *5.2.3.1* and create extra data to compare with the inference results in Subsection *5.2.3.2*, a cross validation was performed. The cross validation was implemented using the Stratified K-fold method. Table 20 contains the average performance (including precision, recall, and F1-score) and standard deviation results obtained using 10 folds (i.e., $K = 10$ with 90/10% train/test splits).

Table 20 – Cross validation average performance results of the test phase

Model	**Precision**	**Recall**	**F1-score**
LGBM	0.99892292 (±0.00148)	0.99893171 (±0.00146)	0.99845948 (±0.00212)
RF	0.99855035 (±0.00228)	0.99852330 (±0.00239)	0.99789003 (±0.00337)
XGB	0.99754671 (±0.00442)	0.99753637 (±0.00442)	0.99642809 (±0.00646)
AdaBoost	0.99698574 (±0.00429)	0.99664981 (±0.00461)	0.99631914 (±0.00632)
HGB	0.99704642 (±0.00449)	0.99707358 (±0.00450)	0.99577591 (±0.00659)
MLP	0.99621339 (±0.00634)	0.99628071 (±0.00636)	0.99455854 (±0.00935)
DT	0.99613499 (±0.00493)	0.99617142 (±0.00490)	0.99444911 (±0.00720)
SVC	0.99596533 (±0.00658)	0.99594951 (±0.00653)	0.99433717 (±0.00968)
LR	0.99554671 (±0.00645)	0.99358971 (±0.00669)	0.99402424 (±0.00954)
Voting	0.99568200 (±0.00641)	0.99558635 (±0.00633)	0.99389705 (±0.00945)
Stacking	0.99544737 (±0.00629)	0.99527674 (±0.00619)	0.99341005 (±0.00923)

Source: Created by the author (2025).

Contrasting the results from the test phase (outlined in Subsection *5.2.3.1*), the Voting and Stacking models exhibited the lowest performance, achieving F1-scores of approximately 99.34% and 99.39%, respectively. Consistent with the test phase results, the linear models (LR and SVC) achieved F1-scores of approximately 99.40% and 99.44%, respectively. Furthermore, LGBM, RF and XGB maintained their position as the top three best models, attaining F1-scores of approximately 99.85%, 99.79%, and 99.64%, respectively. It is noteworthy that all models consistently demonstrated performance levels exceeding 99% across cross validation, with with standard deviations below 1%. This behavior along with the results previously analyzed in Subsections *5.2.3.1* (Training) and *5.2.3.2* (Inference), indicates a model overfitting to the training data.

To further elaborate on the results of the training phase presented in Subsection *5.2.3.1*, the training and classification times of each model were also measured during the cross validation, with their average and standard deviation summarized in Table 21. The ranking remained fairly consistent with the training results, with DT and LR identified as the fastest models, while Stacking was the slowest. In comparison to the inference results discussed in Subsection *5.2.3.2*, the classifier ranking also remained consistent, with DT again recognized as the fastest model, followed by the linear models, while Voting was the slowest. Thus, even though the amount of data used to train and test the models

Table 21 – Cross validation average time results of the test phase

Model	**Average Train Time (s)**	**Model**	**Average Classification Time (s)**
DT	41.7879 (±4.1971)	DT	0.5090 (±0.4237)
LR	125.7473 (±7.2257)	LR	0.5838 (±0.3461)
SVC	340.4176 (±57.1157)	SVC	0.5946 (±0.4041)
XGB	616.3800 (±22.4426)	MLP	1.7114 (±18.1521)
HGB	752.7511 (±80.6718)	XGB	5.4557 (±4.9903)
MLP	885.5553 (±183.9418)	RF	5.9906 (±4.7568)
LGBM	1310.9774 (±13.6833)	Stacking	12.7450 (±10.2109)
AdaBoost	1335.3539 (±109.4027)	LGBM	14.2706 (±12.1558)
Voting	1462.3592 (±71.7484)	HGB	18.7169 (±24.1268)
RF	1719.6241 (±85.3389)	AdaBoost	23.5223 (±0.9397)
Stacking	9078.8962 (±361.6314)	Voting	24.6726 (±25.0216)

Source: Created by the author (2025).

has changed (90%/10% in the cross validation compared to 70%/30% in the training), the results from the cross validation are relatively consistent with those obtained from both the training and inference phases, corroborating that the models suffered overfitting.

5.3 DISCUSSION

The literature review presented in Section 3.2 indicates that classification techniques are prevalent in the context of 5G networks. However, as noted in (103) and the reviewed literature, the application of classifiers to categorize devices across various 5G service axes remains an unresolved issue. Furthermore, as demonstrated in (90) and (100), classification tasks — such as identifying the appropriate service axis for a given UE — may be too complex for a single ML model to handle effectively. In this context, following the suggestion of the authors of (90) to employ four models in a HAC and a voting system could enhance classification performance, and the supervised learning models found on the literature review, the ensemble models AdaBoost, Stacking and Voting were implemented and tested. The findings in Subsection **5.2.3**, particularly those from Subsection *5.2.3.2*, support the notion that a model may excel in detecting one class while struggling with others. Despite the application of multiple models, the mMTC class could not be accurately detected. This highlights the necessity for further exploration of feature selection methods that specifically target the identification of sparse data patterns in mMTC scenarios. This necessity is reinforced by the results in Subsections **5.1.3** and **5.2.3**, which indicate that the eMBB and URLLC datasets exhibit statistically different distributions, while the mMTC dataset does not; nevertheless, the models successfully identified eMBB and URLLC packets. Additionally, (100) suggests investigating time series data generated

by packet traffic, a suggestion supported by the feature importance results presented in Subsection **5.2.2**, while (101) recommends utilizing unsupervised learning methods as a means to differentiate among service axes in UE classification.

Another perspective that may elucidate the results in Subsection **5.2.3** is the impact of user interactions on model inference performance, as discussed in (87) and (100). These interactions can disrupt transmission frequencies, complicating the model's ability to extract patterns or generalize effectively. The study (100) illustrates the challenges in generalizing patterns from datasets that include traffic from smart temperature sensors and smartwatches, where user behavior significantly influences traffic patterns. The results indicate that the accuracy of detecting smartwatches was nearly 0%, regardless of the dataset utilized. Consequently, the authors assert that device classification remains an open problem within the field of Computer Networks and recommend future investigations into service-aware scenarios, where devices are classified based on the services they utilize.

As detailed in Section 4.3, the training dataset was heavily unbalanced, comprising 10 million samples for eMBB and URLLC classes compared to only 1 million for mMTC. The results presented in Subsection **5.2.3** were obtained, as outlined in Section **4.5.2**, after balancing the data through SMOTE. Despite this balancing, the models demonstrated signs of overfitting. Therefore, conducting a detailed feature analysis (e.g., using the Pearson Correlation Coefficient (142), as in (143)) may reveal relationships between features, facilitating feature reduction or the addition of new features to benefit model learning. As suggested in (133), automated hyperparameter tuning (e.g., via Grid Search, as in (137)) could be beneficial, either individually or in combination with the aforementioned methods.

It is important to note that, as described in Subsection **5.1.1**, traffic captures occurred at the UPF, which differs from the data collection methods used to create the training datasets (110) and (114). This discrepancy likely influenced model inference performance, as evidenced by the results in Subsection *5.2.3.2*. Furthermore, the occurrence frequency of protocols varies significantly between the training and inference datasets, as shown in Subsection *5.1.2.1*. For instance, the eMBB class includes PNIO in the inference dataset, which was absent in the training data. This variation can be attributed to both the locations of the captures (e.g., in the UE or in the 5GC's UPF) and the evolution of services generating the traffic over time (e.g., newer protocols being adopted), as the inference dataset was created in 2025, while the training datasets (110) and (114) were compiled in 2022 and 2021, respectively. The results in Subsection **5.1.3** also corroborate that the difference between the distributions of the training and inference data in the eMBB and URLLC were statistically significant. Given the findings in Subsection **5.2.2**, which highlight the importance of the inter packet arrival time (`Time_delta`), it is plausible that model performance was adversely affected due to the differences in the traffic generator implementation in the mMTC scenario. Additionally, these results suggest that model retraining is essential to maintain high levels of classification performance, due to the

expected changes in the services transmitting packets through the 5GC.

Regarding the training and classification times discussed in Subsection **5.2.3**, DT consistently emerged as the fastest model for these tasks. While DT demonstrated superior performance during training and cross validation, it was significantly outperformed by AdaBoost in mMTC burst classification (0% vs. approximately 99.9% F1-score). Therefore, if training and classification times are prioritized — such as in dynamic environments requiring frequent model retraining and high inference rates due to numerous devices — DT might be the preferred model. However, besides achieving commendable classification performance in eMBB and URLLC classes, DT, like the other models, failed to correctly classify packets in the mMTC capture scenarios. Notably, the model raw performance results are publicly available on Zenodo (39).

Finally, it is crucial to emphasize that integrating the NWDAF with the SBI of the 5GC, rather than operating in offline mode, would facilitate automatic data acquisition from network packets. This integration would streamline the processing steps outlined in Section 4.1, enable the generation of classification results as presented in Subsection **5.2.3**, and support decision-making processes that influence the operation of the 5GS, such as RAN transmission frequency adaptation as in (10), slicing reconfiguration as in (11) and (12), and UPF selection as in (13).

5.4 SUMMARY

This chapter presented the experimental research results. The packet capture executed to generate the 5G inference dataset was also presented. A statistical frequency analysis of two features (Protocol Labels and Frame Length) along with hypothesis tests on the features identified as significant by DT and RF, was conducted to compare the training dataset with the inference dataset, thereby supporting the discussion on model performance results. The performance of the models was evaluated across three distinct stages: training, inference, and cross validation, with accompanying discussions on the obtained evaluation results.

Notably, the DT model demonstrated exceptional classification time performance, achieving approximately 23 seconds of model training time and an average of 42 ms during cross validation. It recorded around 29 ms for inference (excluding the mMTC probabilistic capture, where it achieved approximately 0.7 ms), making it the fastest model while maintaining F1-scores exceeding 99% in both the test and cross validation phases, and, respectively, 99.80% and 99.71% in eMBB and URLLC inferences, thus exhibiting commendable classification performance. In contrast, AdaBoost was the only model capable of detecting the mMTC packets in the burst scenario. Besides also achieving over 99.8% F1-score in the eMBB inference, it completely failed to detect the URLLC packets.

6 CONCLUSION

Following the selection of two real-world 5G traffic datasets, experiments were conducted with eleven distinct ML models within an implemented NWDAF-based functionality, demonstrating the classification of UEs based on network traffic observed in the environment, while also conducting a model performance evaluation. Additionally, this work successfully achieved the development of a 5G simulated experimental environment utilizing the FAD tool, alongside the creation of a 5G simulated traffic dataset generated by a custom traffic generator.

The experimental results indicated that while the DT model was the fastest classifier in terms of training, cross validation, and inference, its performance during inference was significantly influenced by the the training data used, making it incapable of accurately classifying mMTC packets in the inference dataset. In contrast, AdaBoost successfully detected eMBB and mMTC packets in the burst inference scenario; however it failed to do so in the URLLC and mMTC probabilistic scenarios. Notably, the mMTC class presented the greatest challenge for the models to learn under the current conditions. As discussed through the Chapter 5, this difficulty arose from differences in the capture environments, the implementation of the traffic generator in the mMTC probabilistic capture, and the differences in the distributions of mMTC captures for the analyzed features. Despite these challenges, which may necessitate retraining, the use of the models remains promising due to the performance results presented in Subsection **5.2.3**.

6.1 CONTRIBUTIONS

This work has made scientific and technical contributions to the Computer Networks field focusing on data analytics for UE classification within 5G networks. Through a comprehensive literature review, the current state of the art in UE classification was investigated focusing on open science aspects, thereby identifying the need to evaluate using ensemble learning models. Moreover, a publicly available 5G simulated traffic capture dataset comprising three different types of traffic was generated, facilitating further experimentation and validation in the domain.

Furthermore, it was possible to support the open science technical landscape by implementing a 5G simulated testing environment using the FLOSS FAD tool, alongside the development of network traffic generation prototypes, implementing of NWDAF-like functionality in the form of a ML pipeline to classify UEs according to their network traffic. The release of the source code, dataset (in the mode detailed PCAP format), and supporting documentation not only improves the reproducibility of the obtained results but also serves as a valuable resource for researchers and practitioners in the field.

6.2 LIMITATIONS

The results achieved by the machine learning models indicate that accurate detection of the mMTC class necessitates specific data augmentation or feature engineering techniques, along with more comprehensive model tuning.

Additionally, the designed pipeline operates solely in an offline mode, requiring prior capture of the packets utilized for training. The NWDAF functionality remains to be implemented as a 5G NF integrated into the 5GC, which is essential for enabling the use of inference results by other consumer NFs.

Moreover, the file preprocessing functions utilized by `pcap_extract.sh` and `export_JSON.py` exhibit high RAM usage, which limits task parallelization and necessitates substantial memory resources for execution.

Furthermore, the implemented solutions have not been validated in environments with a higher number of devices, which may affect the volume of data available for classification. This limitation also necessitates the inclusion of additional applications in the training dataset to encompass a broader range of use cases. In a production environment, as discussed in (87) and (100), user interactions are likely to influence the level of noise present in the data. The variability and volume of data are expected to increase due to a greater number of connected devices and the execution of multiple applications by user equipment, particularly in networks with higher transmission speeds.

It is worth noting that the inference dataset was created using simulated UEs due to the availability of realistic simulators and the challenges involved in configuring a physical 5G test environment. Thus, the performance results of the model inference are expected to change depending on the dataset utilized.

6.3 FUTURE WORK

This work establishes a foundation for multiple future research directions, which can be categorized into three primary areas: computer networks experimentation focusing on advancements in the field of computer networks; Machine Learning performance aimed at enhancing model performance; and 5G data analytics, which seeks to improve technical contributions in alignment with 3GPP specifications and recommendations.

Future experiments should focus on deploying a 5GC instance in a cloud environment or a geographically distant network from the UE to better reflect real-world conditions influenced by Internet traffic. It is also advisable to incorporate COTS devices alongside simulated ones and to conduct packet captures in various elements of the environment, such as the UE, gNB, or DN, to assess their impact on classification outcomes. Additionally, including non-3GPP networks like Wi-Fi in the experimental setup would allow for an examination of the proposed pipeline's ability to identify these networks.

Moreover, extending the experiments to utilize a training dataset with real mMTC traffic, rather than arbitrarily generated data, is crucial. Creating a dataset with PCAP formatted traffic from a larger number of devices across the three classes (eMBB, URLLC, and mMTC) would strengthen the findings. A more in-depth analysis of additional characteristics, along with redesigning the `box-plotter.py` script to incorporate scale adjustments and feature comparisons across different captures, would enhance the understanding of model learning processes and improve result explainability. Finally, it is recommended to implement the traffic generator as a separate module, expanding its capabilities to simulate background traffic and, as suggested in (144), generate 5G traffic based on stochastic processes and mathematical functions. To enhance model performance during inference, as discussed in Section 5.3, it is recommended to implement automated hyperparameter optimization, apply undersampling techniques, consider including additional features or feature reduction methods, and explore unsupervised learning approaches.

Integrating the NWDAF functionality implementing an event-based publish-subscribe scheme (145) as a 5GC NF, would enhance data acquisition from other NFs, allowing the ML pipeline to operate alongside normal 5GC execution. It would also facilitate the transmission of classification results for decision-making, and support the development of a data analytics consumer NF. Future work might also focus on implementing abnormal UE detection and incorporating additional KPIs or features in line with 3GPP Release 18 specifications (61).

REFERENCES

1 SAGAN, Carl. **The Demon-Haunted World**: science as a candle in the dark. London: Headline Book Publishing, 1997. 438 p.

2 TELECO. *In*: Teleco. **Market Share das Operadoras de Celular 5G no Brasil**. Teleco: Inteligência em Telecomunicações, 2025. Available from: https://www.teleco.com.br/5g_brasil.asp. Accessed on: 14 feb. 2025.

3 GSMA. *In*: GSMA Newsroom. **5G Momentum Continues with 1.6 Billion Connections Worldwide, Rising to 5.5 Billion by 2030, According to GSMA Intelligence**. GSMA, 2024. Available from: https://www.gsma.com/newsroom/press-release/5g-momentum-continues-with-1-6-billion-connections-worldwide-rising-to-5-5-billion-by-2030-according-to-gsma-intelligence/. Accessed on: 21 nov. 2024.

4 BROWN, Gabriel. **Service-Based Architecture for 5G Core Networks**. Heavy Reading, 2017. Available from: https://www.3g4g.co.uk/5G/. Accessed on: 19 dec. 2024.

5 GHADIALY, Zahid. **Control and User Plane Separation of EPC nodes (CUPS) in 3GPP Release-14**. 2017. Available from: https://blog.3g4g.co.uk/2017/12/control-and-user-plane-separation-of.html. Accessed on: 06 feb. 2025.

6 SCHIMITT, Peter; LANDAIS, Bruno; YANG, Frank Yong. **Control and User Plane Separation of EPC nodes (CUPS)**. 2017. Available from: https://www.3gpp.org/news-events/3gpp-news/cups. Accessed on: 06 feb. 2025.

7 SHAKYA, Joshua; GHRIBI, Chaima; MERGHEM-BOULAHIA, Leila. Agent-based modeling and simulation for 5G and beyond networks: a comprehensive survey. **Simulation Modelling Practice And Theory**, [S.L.], v. 130, p. 102855-102887, jan. 2024

8 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 29.520 version 15.11.0 Release 15**: 5G System — Network Data Analytics Services — Stage 3. Sophia Antipolis: Etsi, 2023. 41 p.

9 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 23.288 version 17.12.0 Release 17**: Architecture enhancements for 5G System (5GS) to support network data analytics services. Sophia Antipolis: Etsi, 2024. 211 p.

10 BARTSIOKAS, Ioannis A.; GKONIS, Panagiotis K.; KAKLAMANI, Dimitra I.; VENIERIS, Iakovos S.. ML-Based Radio Resource Management in 5G and Beyond Networks: a survey. **Ieee Access**, [S.L.], v. 10, p. 83507-83528, 2022. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/access.2022.3196657.

11 RAFIQUE, Wajid; BARAI, Joyeeta Rani; FAPOJUWO, Abraham O.; KRISHNAMURTHY, Diwakar. A Survey on Beyond 5G Network Slicing for Smart Cities Applications. **Ieee Communications Surveys & Tutorials**, [S.L.], v. 27, n. 1, p. 595-628, feb. 2025. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/comst.2024.3410295.

12 LIMANI, Xhulio; TROCH, Arno; CHEN, Chieh-Chun; CHANG, Chia-Yu; GAVRIELIDES, Andreas; CAMELO, Miguel; MARQUEZ-BARJA, Johann M.; SLAMNIK-KRIJEŁTORAC, Nina. Optimizing 5G Network Slicing: an end-to-end approach with isolation principles. **2024 Ieee Conference On Network Function Virtualization And Software Defined Networks (Nfv-Sdn)**, Natal, p. 1-6, 5 nov. 2024.

13 FREE5GC. **Influence Traffic Routing**. 2025. Available from: https://free5gc.org/guide/8-traffic-influence/. Accessed on: 08 may 2025.

14 SULTAN, Alain. *In*: 3GPP Technologies. **5G System Overview**. 3GPP, 2022. Available from: https://www.3gpp.org/technologies/5g-system-overview. Accessed on: 19 dec. 2024.

15 YU, Heejung; LEE, Howon; JEON, Hongbeom. **What is 5G? Emerging 5G Mobile Services and Network Requirements**. Sustainability v. 9, n. 10, p. 1848 , 15 oct. 2017.

16 SERIES, M. **IMT Vision — Framework and overall objectives of the future development of IMT for 2020 and beyond**. Recommendation ITU v. 2083, n. 0, p. 1-22, 2015.

17 ROSIC, Adem. **Evaluating Machine Learning Classifier Approaches, and their Accuracy for the Detection of Cyberattacks on 5G IoT Systems**. [S.l.]: arXiv, 2023. Available from: https://arxiv.org/abs/2311.02317. Accessed on: 13 jan. 2025.

18 THULASIRAMAN, Preetha; HACKETT, Michael; MUSGRAVE, Preston; EDMOND, Ashley; SEVILLE, Jared. Anomaly Detection in a Smart Microgrid System Using Cyber-Analytics: a case study. **Energies**, [S.L.], v. 16, n. 20, p. 7151, 19 oct. 2023.

19 ALQURA'N, Rabee; ALJAMAL, Mahmoud; AL-AIASH, Issa; ALSARHAN, Ayoub; KHASSAWNEH, Bashar; ALJAIDI, Mohammad; ALANAZI, Rakan. Advancing XSS Detection in IoT over 5G: a cutting-edge artificial neural network approach. **Iot**, [S.L.], v. 5, n. 3, p. 478-508, 25 jul. 2024.

20 KIM, Ye-Eun; KIM, Yea-Sul; KIM, Hwankuk. Effective Feature Selection Methods to Detect IoT DDoS Attack in 5G Core Network. **Sensors**, [S.L.], v. 22, n. 10, p. 3819, 18 may 2022.

21 BORGESEN, Michael. Evaluating Variant Deep Learning and Machine Learning Approaches for the Detection of Cyberattacks on the Next Generation 5G Systems. 2020. 49 f. Dissertação (Mestrado) - Curso de Network And Computer Security, College Of Engineering, Suny Polytechnic Institute, New York, 2020.

22 FARRERAS, Miquel; PAILLISSÉ, Jordi; FÀBREGA, Lluís; VILÀ, Pere. Generation of a network slicing dataset: the foundations for ai-based b5g resource management. **Data In Brief**, [S.L.], v. 55, p. 110738, ago. 2024.

23 RADOGLOU-GRAMMATIKIS, Panagiotis; NAKAS, George; AMPONIS, George; GIANNAKIDOU, Sofia; LAGKAS, Thomas; ARGYRIOU, Vasileios; GOUDOS, Sotirios; SARIGIANNIDIS, Panagiotis. 5GCIDS: an intrusion detection system for 5g core with ai and explainability mechanisms. 2023 Ieee Globecom Workshops (Gc Wkshps), [S.L.], p. 353-358, 4 dez. 2023

24 INTERNET ENGINEERING TASK FORCE. **DRAFT-IETF-OPSAWG-PCAP-05**: PCAP Capture File Format. 5 ed. [S. L.]: Ietf, 2025. 5 p. Available from: https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/05/. Accessed on: 22 may 2025.

25 WIRESHARK. **Libpcap File Format**. 2020. Available from: https://wiki.wireshark.org/Development/LibpcapFileFormat. Accessed on: 22 may 2025.

26 OLIVEIRA, Leonardo Azalim de; SILVA, Edelberto Franco. Estudo e Avaliação de Métodos de Autenticação EAP na Infraestrutura de Redes de Telecomunicação 5G. **Anais Estendidos do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (Sbseg Estendido 2023)**, [Juiz de Fora], p. 97-100, 18 sep. 2023.

27 FREE5GC (Taiwan). **free5GC**: open source 5g core network based on 3gpp r15 . Open source 5G core network based on 3GPP R15. 2025. Available from: https://free5gc.org/. Accessed on: 19 mar. 2025.

28 GÜNGÖR, Ali. UERANSIM: open source 5g ue and ran (gnodeb) implementation. Open source 5G UE and RAN (gNodeB) implementation. 2025. Available from: https://github.com/aligungr/UERANSIM. Accessed on: 17 mar. 2025.

29 OLIVEIRA, Leonardo Azalim de. **FAD**: free5gc auto deploy. free5GC Auto Deploy. 2024. Available from: https://github.com/oliveiraleo/free5gc-auto-deploy. Accessed on: 02 apr. 2025.

30 OLIVEIRA, Leonardo Azalim de; SILVA, Rodrigo Oliveira; LIMA, Pedro Campos; PEREIRA, Antônio Marcos Souza; VALADARES, Júlia Almeida; SILVA, Edelberto Franco; DANTAS, Mário Antônio Ribeiro. Análise da Funcionalidade da NWDAF no Core 5G Sobre um Conjunto de Dados. **Anais do Xlii Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (Sbrc 2024)**, [Niteroi], p. 798-811, 20 may 2024.

31 OLIVEIRA, Leonardo Azalim de; SILVA, Edelberto Franco; DANTAS, Mário Antônio Ribeiro. A NWDAF Study Employing Machine Learning Models on a Simulated 5G Network Dataset. **2024 Ieee Symposium On Computers And Communications (Iscc)**, [Paris], p. 1-6, 26 jun. 2024

32 OLIVEIRA, Leonardo Azalim de; SILVA, Edelberto Franco. Eduroam e 5G: autenticação integrada via redes móveis e wi-fi no core 5g. **Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (Sbseg Estendido 2024)**, [S.L.], p. 189-192, 16 sep. 2024.

33 OLIVEIRA, Leonardo Azalim de; SILVA JUNIOR, Antônio Marcos da; PINTO, Mariana Siano. **Introdução à ambientes de experimentação 5G**. 2024. XXVI Semana da Computação DCC/UFJF. Available from: https://netlab.ice.ufjf.br/courses/5g-practical/. Accessed on: 20 jan. 2025.

34 UNESCO. **UNESCO Recommendation on Open Science**. Paris: United Nations Educational, Scientific And Cultural Organization, 2021. Available from: https://doi.org/10.54677/MNMH8546. Accessed on: 19 jun. 2025.

35 OLIVEIRA, Leonardo. **Free5gc/free5gc - Contributions by oliveiraleo**. 2025. Available from: https://github.com/free5gc/free5gc/issues?q=author:oliveiraleo. Accessed on: 10 jul. 2025.

36 OLIVEIRA, Leonardo. **Pull requests by oliveiraleo - free5gc/free5gc.github.io.** 2025. Available from: https://github.com/free5gc/free5gc.github.io/pulls?q=author:oliveiraleo. Accessed on: 10 jul. 2025.

37 ZHENG, Kaiyuan. **Order of the EAP AKA' attributes**. 2022. Available from: https://github.com/aligungr/UERANSIM/issues/592. Accessed on: 01 jul. 2025.

38 OLIVEIRA, Leonardo. **Running TNGFUE on another subnet**. 2024. Available from: https://forum.free5gc.org/t/running-tngfue-on-another-subnet/2571. Accessed on: 01 jul. 2025.

39 OLIVEIRA, Leonardo. **Archive of Master Thesis Artifacts**: user equipment classification in the 5G core. Traffic Characterization for User Equipment Classification in the 5G Core. 2025. Zenodo. Available from: https://doi.org/10.5281/zenodo.15473395. Accessed on: 20 may 2025.

40 OLIVEIRA, Leonardo Azalim de; SILVA, Edelberto Franco; CHAVES, Luciano Jerez. **5G Traffic Capture Dataset**: user equipment classification in the 5G core. Traffic Characterization for User Equipment Classification in the 5G Core. 2025. Zenodo. Available from: https://doi.org/10.5281/zenodo.15064129. Accessed on: 20 may 2025.

41 OLIVEIRA, Leonardo Azalim de. **NWDAF ML**. 2025. Available from: https://github.com/netlabufjf/nwdaf_ml. Accessed on: 02 apr. 2025.

42 3GPP. *In*: 3GPP Technologies. **Introducing 3GPP**. 3GPP, 2024. Available from: https://www.3gpp.org/about-us/introducing-3gpp. Accessed on: 06 mar. 2025.

43 ITU. **About International Telecommunication Union (ITU)**. ITU, 2025. Available from: https://www.itu.int/en/about/Pages/default.aspx. Accessed on: 10 mar. 2025.

44 ROMANO, Giovanni. *In*: 3GPP Technologies. **3GPP meets IMT-2020**. 3GPP, 2020. Available from: https://www.3gpp.org/technologies/3gpp-meets-imt-2020. Accessed on: 10 mar. 2025.

45 ITU. **IMT-2020**. ITU, 2025. Available from: https://www.itu.int/en/ITU-R/study-groups/rsg5/rwp5d/imt-2020/Pages/default.aspx. Accessed on: 06 mar. 2025.

46 AL-DUJAILI, Mohammed Jawad; AL-DULAIMI, Mohammed Abdulzahra. **Fifth-Generation Telecommunications Technologies: Features, Architecture, Challenges and Solutions**. Wireless Personal Communications v. 128, n. 1, p. 447–469 , jan. 2023.

47 WIKIPEDIA. **1G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/1G. Accessed on: 06 mar. 2025.

48 WIKIPEDIA. **2G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/2G. Accessed on: 06 mar. 2025.

49 WIKIPEDIA. **3G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/3G. Accessed on: 06 mar. 2025.

50 WIKIPEDIA. **4G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/4G. Accessed on: 06 mar. 2025.

51 3GPP. *In*: 3GPP Specifications & Technologies. **Release 15**. 3GPP, 2019. Available from: https://www.3gpp.org/specifications-technologies/releases/release-15. Accessed on: 06 mar. 2025.

52 WIKIPEDIA. **5G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/5G. Accessed on: 06 mar. 2025.

53 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 23.501 version 15.13.0 Release 15**: System architecture for the 5G System (5GS). Sophia Antipolis: Etsi, 2022. 253 p.

54 CHAI, Yu-Herng; LIN, Fuchun Joseph. Evaluating Dedicated Slices of Different Configurations in 5G Core. **Journal Of Computer And Communications**, [S.L.], v. 09, n. 07, p. 55-72, 2021.

55 SDXCENTRAL STUDIOS. **What Is the Radio Access Network (RAN)?**. SDxCentral Studios, 2025. Available from: https://www.sdxcentral.com/5g/ran/definitions/radio-access-network/. Accessed on: 07 mar. 2025.

56 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TR 21.905 version 18.0.0 Release 18**: Vocabulary for 3GPP Specifications. Sophia Antipolis: Etsi, 2024. 69 p.

57 SERIES, I. **Integrated Services Digital Network (ISDN) — General Structure — Vocabulary of terms for ISDNs**. Recommendation ITU v. 112, p. 1-20, 1993.

58 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 29.060 version 17.4.0 Release 17**: GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface. Sophia Antipolis: Etsi, 2022. 199 p.

59 KIM, Ye-Eun; KIM, Min-Gyu; KIM, Hwankuk. **Detecting IoT Botnet in 5G Core Network Using Machine Learning**. Computers, Materials & Continua v. 72, n. 3, p. 4467–4488 , 2022.

60 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 29.575 version 17.3.0 Release 17**: Analytics Data Repository Services — Stage 3. Sophia Antipolis: Etsi, 2023. 51 p.

61 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 23.288 version 18.9.0 Release 18**: Architecture enhancements for 5G System (5GS) to support network data analytics services. Sophia Antipolis: Etsi, 2025. 329 p.

62 NIU, Yuxia; ZHAO, Song; SHE, Xiaoming; CHEN, Peng. A Survey of 3GPP Release 18 on Network Data Analytics Function Management. In: IEEE/CIC INTERNATIONAL CONFERENCE ON COMMUNICATIONS IN CHINA (ICCC

WORKSHOPS), 2022., 2022, Sanshui. **Proceedings [...]** . Sanshui: Ieee, 2022. p. 146-151.

63 3RD GENERATION PARTNERSHIP PROGRAM. **3GPP TS 23.502 version 17.13.0 Release 17**: Procedures for the 5G System (5GS). Sophia Antipolis: Etsi, 2024. 755 p.

64 RUPANAGUNTA, Sriram. **NWDAF Rel 17 Explained — Architecture, Features and Use Cases**. Aarna, 2021. Available from: https://www.aarna.ml/post/nwdaf-rel-17-explained-architecture-features-and-use-cases. Accessed on: 10 mar. 2025.

65 KUAN, Liu H. **NWDAF introduction**. Free5GC, 2024. Available from: https://free5gc.org/blog/20241127/20241127/. Accessed on: 10 mar. 2025.

66 5G AMERICAS. **Becoming 5G-Advanced**: the 3gpp 2025 roadmap. the 3GPP 2025 Roadmap. 2022. Available from: https://www.5gamericas.org/becoming-5g-advanced-the-3gpp-roadmap/. Accessed on: 18 mar. 2025

67 ERICSSON. **5G Advanced**: evolution towards 6g. Evolution towards 6G. 2023. Available from: https://www.ericsson.com/en/reports-and-papers/white-papers/5g-advanced-evolution-towards-6g. Accessed on: 18 mar. 2025.

68 SERIES, M. **Framework and overall objectives of the future development of IMT for 2030 and beyond**. Recommendation ITU v. 2160, n. 0, p. 1-21, 2023.

69 WIKIPEDIA. **6G**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/6G. Accessed on: 18 mar. 2025.

70 WIKIPEDIA. **Machine learning**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/Machine_learning. Accessed on: 20 may 2025.

71 MAVROMATIS, Ioannis; KATSAROS, Kostas; KHAN, Aftab. Computing Within Limits: an empirical study of energy consumption in ml training and inference. **Arxiv Preprint Arxiv:2406.14328**, [S.L.], p. 1-15, 20 jun. 2024. ArXiv. http://dx.doi.org/10.48550/ARXIV.2406.14328.

72 MAAYAN, Gilad David. **A Practical Guide to Working with Testing and Training Data in ML Projects**. 2023. Available from: https://www.computer.org/publications/tech-news/trends/machine-learning-projects-training-testing. Accessed on: 20 mar. 2025.

73 PEDREGOSA, F. et al. **Scikit-learn: Machine Learning in Python**. Journal of Machine Learning Research v. 12, p. 2825-2830 , 2011.

74 LEARN, Scikit. **Metrics and scoring: quantifying the quality of predictions**. Website, 2024. Available from: https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score. Accessed on: 06 mar. 2025

75 ZAKI, Mohammed J.; MEIRA JUNIOR, Wagner. **Data Mining and Machine Learning**: fundamental concepts and algorithms. 2. ed. Cambridge: Cambridge University Press, 2020. 777 p.

76 WIKIPEDIA. **Receiver operating characteristic**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/Receiver_operating_characteristic. Accessed on: 20 mar. 2025.

77 SCIKIT-LEARN. **Multiclass Receiver Operating Characteristic (ROC)**: one-vs-one multiclass roc. One-vs-One multiclass ROC. 2007. Available from: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#one-vs-one-multiclass-roc. Accessed on: 20 mar. 2025.

78 PATI, D.; LORUSSO, L. N. **How to Write a Systematic Review of the Literature**. HERD: Health Environments Research & Design Journal, v. 11, n. 1, p. 15–30, 28 dec. 2018.

79 LEVY, Yair; J. ELLIS, Timothy. **A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research**. Informing Science: The International Journal of an Emerging Transdiscipline v. 9, p. 181–212 , 2006.

80 CARRERA-RIVERA, Angela et al. **How-to conduct a systematic literature review**: A quick guide for computer science research. MethodsX v. 9, p. 101895 , 2022.

81 HARZING, Anne-Wil. **Publish or Perish**. Software, 2007. Available from: https://harzing.com/resources/publish-or-perish. Accessed on: 29 oct. 2024

82 FEDER, Alexander. **BibTeX**. Software, 2006. Available from: https://www.bibtex.org/. Accessed on: 26 feb. 2025

83 FREITAS, V.; SEGATTO, W. **Parsifal**. Software, 2018. Available from: https://parsif.al. Accessed on: 29 oct. 2024

84 Wikipedia. **Qualis (CAPES)**. Website, 2024. Available from: https://en.wikipedia.org/wiki/Qualis_(CAPES). Accessed on: 08 nov. 2024

85 Elsevier. **CiteScore metrics you can verify and trust**. Website, 2024. Available from: https://www.elsevier.com/products/scopus/metrics/citescore. Accessed on: 08 nov. 2024

86 EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH ([Switzerland]). Cern (org.). **Zenodo**. 2013. Available from: https://www.zenodo.org/. Accessed on: 07 apr. 2025.

87 BARTOLEC, Ivan; ORSOLIC, Irena; SKORIN-KAPOV, Lea. **Impact of User Playback Interactions on In-Network Estimation of Video Streaming Performance**. IEEE Transactions on Network and Service Management v. 19, n. 3, p. 3547–3561 , set. 2022.

88 PAOLINI, Emilio et al. **Real-Time Network Packet Classification Exploiting Computer Vision Architectures**. IEEE Open Journal of the Communications Society v. 5, p. 1155–1166 , 2024.

89 SAMARAKOON, Sehan et al. **5G-NIDD: A Comprehensive Network Intrusion Detection Dataset Generated over 5G Wireless Network**. IEEE DataPort, 2022. DOI:10.21227/xtep-hv36.

90 KOURSIOUMPAS, Nikolaos et al. **AI-driven, Context-Aware Profiling for 5G and Beyond Networks**. IEEE Transactions on Network and Service Management v. 19, n. 2, p. 1036–1048 , jun. 2022.

91 SEVGICAN, Salih et al. **Intelligent network data analytics function in 5G cellular networks using machine learning**. Journal of Communications and Networks v. 22, n. 3, p. 269–280 , jun. 2020.

92 DA SILVA, Douglas Chagas et al. **A Novel Approach to Multi-Provider Network Slice Selector for 5G and Future Communication Systems**. Sensors v. 22, n. 16, p. 6066 , 13 ago. 2022.

93 RIZWAN, Ali et al. **A Zero-Touch Network Service Management Approach Using AI-Enabled CDR Analysis**. IEEE Access v. 9, p. 157699–157714 , 2021.

94 ANFAR, Mohamad Rimas Mohamad; MWANGAMA, Joyce. **Machine Learning-Based Service Differentiation in the 5G Core Network**. In: 2021 INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE IN INFORMATION AND COMMUNICATION (ICAIIC), 13 abr. 2021, Jeju Island, Korea (South). IEEE, 13 abr. 2021. p.144–149. 978-1-72817-638-3. .

95 FRAUNHOFER FOKUS (Germany). **Open5GCore**. Available from: https://www.open5gcore.org/. Accessed on: 15 mar. 2025.

96 GUERRA-MANZANARES, Alejandro; MEDINA-GALINDO, Jorge; BAHSI, Hayretdin; NÕMM, Sven. MedBIoT: generation of an iot botnet dataset in a medium-sized iot network. **Proceedings Of The 6th International Conference On Information Systems Security And Privacy**, Valletta, v. 1, p. 207-218, 2020.

97 MOHAMMEDALI, Noor Abdalkarem et al. **Enhancing Service Classification for Network Slicing in 5G Using Machine Learning Algorithms**. In: AL-BAKRY, Abbas M. et al. (Orgs.). . New Trends in Information and Communications Technology Applications. Communications in Computer and Information Science. Cham: Springer Nature Switzerland, 2023. 1764 v. p. 25–37. 978-3-031-35441-0.

98 MATHWORKS (United States). MATLAB. 2025. Available from: https://www.mathworks.com/products/matlab.html. Accessed on: 19 mar. 2025.

99 WIRESHARK FOUNDATION (United States). **Wireshark**: the world's most popular network protocol analyzer. The world's most popular network protocol analyzer. 1998. Available from: https://www.wireshark.org/. Accessed on: 19 mar. 2025.

100 TAKASAKI, Chikako et al. **Device Type Classification Based on Two-Stage Traffic Behavior Analysis**. IEICE Transactions on Communications v. E107.B, n. 1, p. 117–125 , 1 jan. 2024.

101 MANIAS, Dimitrios Michael; CHOUMAN, Ali; SHAMI, Abdallah. **An NWDAF Approach to 5G Core Network Signaling Traffic: Analysis and Characterization**. In: GLOBECOM 2022 - 2022 IEEE GLOBAL COMMUNICATIONS CONFERENCE, 4 dez. 2022, Rio de Janeiro, Brazil. IEEE, 4 dec. 2022. p.6001–6006. 978-1-66543-540-6. .

102 SUKCHAN LEE (South Korea). **Open5GS**: open source implementation for 5g core and epc. Open Source implementation for 5G Core and EPC. 2024. Available from: https://open5gs.org/. Accessed on: 17 mar. 2025.

103 SILVA, Gabriel Henrique Davanço. **Classificação de tráfego por classes de serviço no núcleo 5G**. 2022.

104 DAHMEN-LHUISSIER, Sabine. **Zero touch network & Service Management (ZSM)**. 2025. Available from: https://www.etsi.org/technologies/zero-touch-network-service-management. Accessed on: 15 jul. 2025.

105 SOUZA NETO, Natal V. et al. **Evolved NWDAF Towards a Fully Distributed Artificial Intelligence in the 6G Network Architecture**. Anais do IV Workshop de Redes 6G, p. 15-25 , 2024.

106 RISCHKE, Justus; SOSSALLA, Peter; ITTING, Sebastian; FITZEK, Frank H. P.; REISSLEIN, Martin. 5G Campus Networks: a first measurement study. **Ieee Access**, [S.L.], v. 9, p. 121786-121803, 2021.

107 MQTT. **MQTT**: the standard for iot messaging. The Standard for IoT Messaging. 2024. Available from: https://mqtt.org/. Accessed on: 01 jul. 2025.

108 ENGLISH, John. **Service Assurance Questions in the Time of 5G Standalone**: how does service assurance change with the movement to 5g standalone?. How does service assurance change with the movement to 5G standalone?. 2023. Available from: https://www.netscout.com/blog/service-assurance-questions-time-5g-standalone. Accessed on: 02 apr. 2025.

109 IEEE (United States Of America). **IEEE DataPort**: dataset storage and dataset search platform. Dataset Storage and Dataset Search Platform. 2025. Available from: https://ieee-dataport.org/. Accessed on: 28 mar. 2025.

110 CHOI, Yong-Hoon; KIM, Daegyeom; KO, Myeongjin. **5G traffic datasets**. IEEE DataPort, 2023. DOI:10.21227/ewhk-n061.

111 GOOGLE (United States Of America). Google Scholar. 2025. Available from: https://scholar.google.com/. Accessed on: 28 may. 2025.

112 KAGGLE (United States Of America). **Kaggle**: your machine learning and data science community. Your Machine Learning and Data Science Community. 2025. Available from: https://www.kaggle.com/. Accessed on: 28 mar. 2025.

113 DATA. [Basel], 2016. Available from: https://www.mdpi.com/journal/data. Accessed on: 28 mar. 2025.

114 RISCHKE, Justus. **5G campus networks**: measurement traces. IEEE DataPort, 2021. DOI:10.21227/xe3c-e968.

115 EMMERICH, Paul; GALLENMÜLLER, Sebastian; RAUMER, Daniel; WOHLFART, Florian; CARLE, Georg. MoonGen. **Proceedings Of The 2015 Internet Measurement Conference**, [Tokyo], p. 275-287, 28 oct. 2015.

116 STACK OVERFLOW (United States). **Stack Overflow Developer Survey 2022**. 2022. Available from: https://survey.stackoverflow.co/2022/#section-version-control-version-control-platforms. Accessed on: 01 jul. 2025.

117 FREE5GC (Taiwan). User Guide. [2020]. Available from: https://free5gc.org/guide/. Accessed on: 02 apr. 2024.

118 OLIVEIRA, Leonardo. **Running TNGFUE on another subnet**. 2024. Available from: https://forum.free5gc.org/t/running-tngfue-on-another-subnet/2571. Accessed on: 01 jul. 2025.

119 OLIVEIRA, Leonardo. **Free5gc/free5gc**: contributions by oliveiraleo. Contributions by oliveiraleo. 2024. Available from: https://github.com/free5gc/free5gc/issues?q=author:oliveiraleo. Accessed on: 01 jul. 2025.

120 OLIVEIRA, Leonardo. **Feat: Improve TNGFUE execution flow by oliveiraleo - Pull Request #2 - free5gc/tngfue**. 2024. Available from: https://github.com/free5gc/tngfue/pull/2. Accessed on: 01 jul. 2025.

121 OLIVEIRA, Leonardo. **Pull requests by oliveiraleo - free5gc/free5gc.github.io**. 2024. Available from: https://github.com/free5gc/free5gc.github.io/pulls?q=is:pr+author:oliveiraleo. Accessed on: 01 jul. 2025.

122 OLIVEIRA, Leonardo Azalim de. **PCAP-dataExtractor**: Python code to parse JSON network traffic data to CSV file. 2025. Available from: https://github.com/oliveiraleo/PCAP-dataExtractor. Accessed on: 02 apr. 2025.

123 SIVANATHAN, Arunan et al. **Characterizing and classifying IoT traffic in smart cities and campuses**. In: 2017 IEEE CONFERENCE ON COMPUTER COMMUNICATIONS: WORKSHOPS (INFOCOM WKSHPS), maio 2017, Atlanta, GA. Anais... Atlanta, GA: IEEE, maio 2017. p.559–564. 978-1-5386-2784-6. 2017.

124 WIRESHARK. Tshark: dump and analyze network traffic. Dump and analyze network traffic. [2025]. Available from: https://www.wireshark.org/docs/man-pages/tshark.html. Accessed on: 02 apr. 2025.

125 WIRESHARK. **PROFINET IO (PN-IO)**. 2020. Available from: https://wiki.wireshark.org/PROFINET/IO. Accessed on: 09 apr. 2025.

126 DREIBHOLZ, Thomas. High-Precision Round-Trip Time Measurements in the Internet with HiPerConTracer. **2023 International Conference On Software, Telecommunications And Computer Networks (Softcom)**, [S.L.], p. 1-7, 21 sep. 2023.

127 WIRESHARK. **SINEC H1 (H1)**. 2020. Available from: https://wiki.wireshark.org/H1. Accessed on: 09 apr. 2025.

128 INTERNET ENGINEERING TASK FORCE. RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport. [S. L.], 2021. Available from: https://datatracker.ietf.org/doc/html/rfc9000. Accessed on: 10 apr. 2025.

129 RIKITAKE, Kenji. Do not enable QUIC on 1280-byte MTU IPv6 networks. 2024. Available from: https://gist.github.com/jj1bdx/1adac3e305d0fb6dee90dd5b909513ed. Accessed on: 10 apr. 2025.

130 BROWNLEE, Jason. **A Gentle Introduction to the Chi-Squared Test for Machine Learning**. 2019. Available from: https://machinelearningmastery.com/chi-squared-test-for-machine-learning/. Accessed on: 01 jul. 2025.

131 CORDER, Gregory W.; FOREMAN, Dale I.. Nonparametric statistics: a step-by-step approach. 2. ed. Nashville: John Wiley & Sons, 2014. 283 p. ISBN: 978-1-118-84031-3.

132 MANN, H. B.; WHITNEY, D. R.. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. **The Annals Of Mathematical Statistics**, [S.L.], v. 18, n. 1, p. 50-60, mar. 1947.

133 BROWNLEE, Jason. **Machine learning mastery with Python**: understand your data, create accurate models, and work projects end-to-end. 1.4 [S. L.]: Machine Learning Mastery, 2016. 179 p.

134 PEARSON, Karl. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. **The London, Edinburgh, And Dublin Philosophical Magazine And Journal Of Science**, [S.L.], v. 50, n. 302, p. 157-175, jul. 1900.

135 KALIYADAN, Feroze; KULKARNI, Vinay. Types of variables, descriptive statistics, and sample size. **Indian Dermatology Online Journal**, [S.L.], v. 10, n. 1, p. 82, 2019.

136 SUKUMAR, Hamsini. Continuous vs. discrete vs. categorical axis: what is the difference?. What is the difference?. 2025. Available from: https://inforiver.com/insights/continuous-discrete-categorical-axis-difference/. Accessed on: 01 jul. 2025

137 HOSSAIN, Md. Riyad; TIMMER, Douglas. Machine Learning Model Optimization with Hyper Parameter Tuning Approach. Global Journal Of Computer Science And Technology: D, [S.L.], v. 21, n. 2, p. 31, 2021. Monthly.

138 STORN, Rainer; PRICE, Kenneth. Differential Evolution: a simple and efficient heuristic for global optimization over continuous spaces. **Journal Of Global Optimization**, [S.L.], v. 11, n. 4, p. 341-359, dez. 1997.

139 IBM ([United States Of America]). What is random forest? 2025. Available from: https://www.ibm.com/think/topics/random-forest. Accessed on: 16 apr. 2025.

140 WIKIPEDIA. **Random forest**. Wikipedia, 2025. Available from: https://en.wikipedia.org/wiki/Random_forest. Accessed on: 16 abr. 2025.

141 CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P.. SMOTE: synthetic minority over-sampling technique. **Journal Of Artificial Intelligence Research**, [S.L.], v. 16, p. 321-357, 1 jun. 2002.

142 PEARSON, Karl. Determination of the Coefficient of Correlation. **Science**, [S.L.], v. 30, n. 757, p. 23-25, 2 jul. 1909.

143 OLIVEIRA, Júnia Maísa; MORAIS, César; MACEDO, Daniel; NOGUEIRA, José Marcos. A Comparative Analysis of Feature Selection and Machine Learning Algorithms for Enhanced Anomaly Detection in 5G Core Networks. **2025 Global Information Infrastructure And Networking Symposium (Giis)**, [Dubai], p. 1-6, 25 feb. 2025.

144 LEE, Ian W.C.; FAPOJUWO, Abraham O.. Stochastic processes for computer network traffic modeling. Computer Communications, [S.L.], v. 29, n. 1, p. 1-23, dez. 2005.

145 BRADBURY, Richard. UE Data Collection, Reporting and Exposure. 2022. Updated on 13 May 2023. Available from: https://www.3gpp.org/technologies/ue-data-sa4. Accessed on: 14 may 2025.

146 EVIDENTLY AI. **Accuracy, precision, and recall in multi-class classification**. 2025. Available from: https://www.evidentlyai.com/classification-metrics/multi-class-metrics. Accessed on: 08 aug. 2025.

147 LEARN, Scikit. **Metrics and scoring: quantifying the quality of predictions**. Website, 2024. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html. Accessed on: 06 mar. 2025

148 TIM. **Accuracy always equal to recall**. 2021. Cross Validated. Available from: https://stats.stackexchange.com/questions/523695/accuracy-always-equal-to-recall/523715#523715. Accessed on: 08 aug. 2025.

APPENDIX A – Source of the install-go.sh script

As indicated in Section 4.4, this script is part of the FAD tool. The ready-to-use source code and documentation are available in a public GitHub repository (29).

```
#!/usr/bin/env bash

echo "Welcome to the Go installer script"

# Control variables
GO_LANG_VERSION=1.21.8

###############
# Install Go #
###############
echo "[INFO] Installing Go $GO_LANG_VERSION"
echo "[INFO] Downloading the package from source"
# Install Go
wget -nc https://dl.google.com/go/go$GO_LANG_VERSION.linux-amd64.tar.gz
echo "[INFO] Extracting and installing package contents"
sudo tar -C /usr/local -zxf go$GO_LANG_VERSION.linux-amd64.tar.gz
echo "[INFO] Updating envitonment vars"
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
echo "[INFO] Don't forget to reload the bash env using"
echo "source ~/.bashrc"
sleep 0.1 # wait for the file to be writen
echo "[INFO] Go installation finished"
```

Source: Created by the author (2025).

APPENDIX B – Source of the deploy-free5gc.sh script

As indicated in Section 4.4, this script is part of the FAD tool. The ready-to-use source code and documentation are available in a public GitHub repository (29).

```
#!/usr/bin/env bash

echo "Welcome to the free5GC auto deploy script"

sudo -v # cache credentials
if [ $? == 1 ] # check if credentials were successfully cached
then
    echo "[ERROR] Without root permission, you cannot change the hostname nor install packages"
    exit 1
fi

# Control variables (1 = true, 0 = false)
FREE5GC_STABLE_BRANCH_CONTROL=1 # switch between using the free5GC stable branch or latest nightly
FREE5GC_VERSION=v3.4.4 # select the stable branch tag that will be used by the script
FREE5GC_NIGHTLY_COMMIT=a39de62 # select which commit hash will be used by the script
N3IWF_CONFIGURATION_CONTROL=0 # prepare N3IWF configuration if 1 is set
N3IWF_STABLE_BRANCH_CONTROL=1 # switch between using the N3IWF stable or nightly branch
N3IWF_NIGHTLY_COMMIT=9fe155e # select which commit hash will be used by the script
TNGF_CONFIGURATION_CONTROL=0 # prepare TNGF configuration if 1 is set
FIREWALL_RULES_CONTROL=0 # deletes all firewall rules if 1 is set
UBUNTU_VERSION=20 # Ubuntu version where the script is running
GTP5G_VERSION=v0.9.5 # select the version tag that will be used to clone the GTP-U module

function ver { printf "%03d%03d%03d" $(echo "$1" | tr '.' ' '); } # util. to compare versions

# check the number of parameters
if [ $# -gt 3 ]; then
    echo "[ERROR] Too many parameters given! Check your input and try again"
    exit 2
fi
# check the parameters and set the control vars accordingly
if [ $# -ne 0 ]; then
    while [ $# -gt 0 ]; do
```

```
34      case $1 in
35          -nightly)
36              FREE5GC_STABLE_BRANCH_CONTROL=0
37              echo "[INFO] The nightly branch of free5GC will be
    cloned"
38              ;;
39          -n3iwf)
40              N3IWF_CONFIGURATION_CONTROL=1
41              echo "[INFO] N3IWF will be configured during the
    execution"
42              ;;
43          -n3iwf-nightly)
44              N3IWF_CONFIGURATION_CONTROL=1
45              N3IWF_STABLE_BRANCH_CONTROL=0
46              echo "[INFO] N3IWF will be configured during the
    execution"
47              echo "[INFO] The nightly branch of N3IWF will be cloned"
48              ;;
49          -tngf)
50              # verify if the stable version is set to be deployed and
     if FREE5GC_VERSION >= v3.4.3, else deploy nightly version
51              FREE5GC_VERSION_FLOAT=${FREE5GC_VERSION#?} # stripping
    leading 'v' to compare only digits
52              if [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 1 ]; then
53                  if [ $(ver $FREE5GC_VERSION_FLOAT) -gt $(ver 3.4.2)
    ]; then
54                      TNGF_CONFIGURATION_CONTROL=1
55                      echo "[INFO] TNGF will be configured during the
    execution"
56                  else
57                      echo "[ERROR] free5GC $FREE5GC_VERSION was
    selected, however it doesn't contain TNGF"
58                      echo "[INFO] Please, select any version >= v3
    .4.3 or drop the TNGF parameter"
59                      echo "[INFO] If using nightly version, please,
    put the TNGF parameter after the nightly one"
60                      exit 1
61                  fi
62              elif [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 0 ]; then
63                  TNGF_CONFIGURATION_CONTROL=1
64                  echo "[INFO] TNGF will be configured during the
    execution"
65              fi
66              ;;
67          -reset-firewall)
68              FIREWALL_RULES_CONTROL=1
69              echo "[INFO] Firewall rules will be cleaned during the
```

```
    execution"
70                  ;;
71              # -only-setup-n3iwf)
72              # ;;
73              *)
74                  echo "[ERROR] Some input parameter wasn't found. Check
    your input and try again"
75                  exit 1
76                  ;;
77          esac
78          shift
79      done
80  else
81      echo "[INFO] N3IWF will NOT be configured during the execution"
82      echo "[INFO] TNGF will NOT be configured during the execution"
83      echo "[INFO] Firewall rules will NOT be cleaned during the execution
    "
84  fi
85
86  # confirm if conflicting NFs (N3IWF and TNGF) will be configured at the
    same time
87  if [[ $N3IWF_CONFIGURATION_CONTROL -eq 1 && $TNGF_CONFIGURATION_CONTROL
    -eq 1 ]]; then
88      echo "[WARN] Running N3IWF and TNGF at the same time is not yet
    supported by this tool"
89      read -p "Press ENTER to continue or Ctrl+C to abort now"
90  fi
91
92  # check for incompatible versions between 5GC and GTP-U module
93  GTP5G_VERSION_FLOAT=${GTP5G_VERSION#?} # stripping leading 'v' to
    compare only digits
94  if [[ $FREE5GC_VERSION = "v3.4.4" ]] && [ $(ver $GTP5G_VERSION_FLOAT) -
    lt $(ver 0.9.3) ]; then
95      echo "[ERROR] Running UPF from free5GC v3.4.4 requires gtp5g v0.9.3
    !"
96      echo "[INFO] Please, set GTP5G_VERSION to v0.9.3 or higher"
97      echo "[INFO] Version currently set: $GTP5G_VERSION"
98      exit 3
99  fi
100
101 echo "[INFO] Execution started"
102
103 # check your go installation
104 go version
105 echo "[INFO] Go should have been previously installed, if not abort the
    execution"
106 echo "[INFO] The message above must not show a \"command not found\"
```

```
    error"
read -p "Press ENTER to continue or Ctrl+C to abort now"

# Hostname update
echo "[INFO] Updating the hostname"
sudo sed -i "1s/.*/free5gc/" /etc/hostname
HOSTS_LINE=$(grep -n '127.0.1.1' /etc/hosts | awk -F: '{print $1}' -)
sudo sed -i ""$HOSTS_LINE"s/.*/127.0.1.1 free5gc/" /etc/hosts

echo "[INFO] Updating the package database and installing system updates
    "
sudo apt update && sudo apt upgrade -y

# check Ubuntu version
lsb_release -sr | grep "^22"  >/dev/null 2>&1
if [[ $? -eq 0 ]]; then
    echo "[INFO] Ubuntu 22.04 LTS detected. Adjusting the database
    installation accordingly"
    UBUNTU_VERSION=22
elif [[ $? -eq 1 ]]; then
    echo "[INFO] Ubuntu 20.04 LTS detected, continuing..."
    UBUNTU_VERSION=20
else
    echo "[ERROR] Script failed to set UBUNTU_VERSION variable or a
    unsuported version is being used"
    exit 1
fi

# Install CP supporting packages
echo "[INFO] Installing DB"
if [[ $UBUNTU_VERSION -eq 20 ]]; then
    sudo apt -y install mongodb wget git
    sudo systemctl start mongodb
elif [[ $UBUNTU_VERSION -eq 22 ]]; then
    sudo apt -y install gnupg curl
    curl -fsSL https://pgp.mongodb.com/server-7.0.asc | \
       sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor
    echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
    server-7.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-
    org/7.0 multiverse" | \
        sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list
    sudo apt update
    sudo apt install -y mongodb-org
    sudo systemctl start mongod
else
    echo "[ERROR] Script failed to setup the data base"
    exit 1
```

```
148 fi
149
150 # Install UPF supporting packages
151 echo "[INFO] Installing UPF prerequisites"
152 sudo apt -y install gcc g++ cmake autoconf libtool pkg-config libmnl-dev
        libyaml-dev
153 echo "[INFO] Done"
154
155 ######################
156 # Configure host OS #
157 ######################
158 echo "[INFO] Configuring host OS"
159 ip a
160 echo ""
161 echo "Please, enter the 5GC's DN interface name (e.g. the interface that
        has internet access)"
162 echo -n "> "
163 read IFACENAME
164
165 echo "[INFO] Using $IFACENAME as interface name"
166
167 # warn the user before deleting the rules
168 if [ $FIREWALL_RULES_CONTROL -eq 1 ]; then
169     # start to delete old rules
170     echo -n "[INFO] Removing all iptables rules, if any... "
171     sudo iptables -P INPUT ACCEPT
172     sudo iptables -P FORWARD ACCEPT
173     sudo iptables -P OUTPUT ACCEPT
174     sudo iptables -t nat -F
175     sudo iptables -t mangle -F
176     sudo iptables -F
177     sudo iptables -X
178     echo "[OK]"
179 fi
180 echo -n "[INFO] Applying free5GC iptables rules... "
181 sudo iptables -t nat -A POSTROUTING -o $IFACENAME -j MASQUERADE
182 sudo iptables -A FORWARD -p tcp -m tcp --tcp-flags SYN,RST SYN -j TCPMSS
        --set-mss 1400
183 sudo iptables -I FORWARD 1 -j ACCEPT
184 echo "[OK]"
185 echo -n "[INFO] Setting kernel net.ipv4.ip_forward flag... "
186 sudo sysctl -w net.ipv4.ip_forward=1 >/dev/null
187 echo "[OK]"
188 echo -n "[INFO] Stopping and disabling the ufw firewall... "
189 sudo systemctl stop ufw
190 sudo systemctl disable ufw >/dev/null 2>&1
191 echo "[OK]"
```

```
192
193 ########################
194 # Install free5GC's CP #
195 ########################
196 echo "[INFO] Installing the 5GC"
197 if [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 1 ]; then
198     echo "[INFO] Cloning free5GC stable branch"
199     echo "[INFO] Tag/release: $FREE5GC_VERSION"
200     if [[ $FREE5GC_VERSION = "v3.3.0" ]]; then
201         echo "[WARN] Using an older release should be avoided"
202         # v3.3.0
203         git clone -c advice.detachedHead=false --recursive -b
    $FREE5GC_VERSION -j `nproc` https://github.com/free5gc/free5gc.git #
    clones the previous stable build
204         cd free5gc
205         sudo corepack enable # necessary to build webconsole on free5GC
    v3.3.0
206         # Useful script
207         echo "[INFO] Downloading reload_host_config script from source"
208         curl -LOSs https://raw.githubusercontent.com/free5gc/free5gc/
    main/reload_host_config.sh
209
210     elif [[ $FREE5GC_VERSION = "v3.4.1" || $FREE5GC_VERSION = "v3.4.2"
    || $FREE5GC_VERSION = "v3.4.3" || $FREE5GC_VERSION = "v3.4.4" ]];
    then
211         # v3.4.x
212         git clone -c advice.detachedHead=false --recursive -b
    $FREE5GC_VERSION -j `nproc` https://github.com/free5gc/free5gc.git #
    clones the stable build
213         cd free5gc
214     else
215         echo "[ERROR] Script failed to set FREE5GC_VERSION variable" #
    check your spelling, you must keep the "v" (e.g. v.3.4.1 and up)
216         exit 1
217     fi
218 elif [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 0 ]; then
219     echo "[INFO] Cloning free5GC nightly branch"
220     echo "[INFO] Commit: $FREE5GC_NIGHTLY_COMMIT"
221     echo "[WARN] Unless you know what you are doing, using the nightly
    branch should be avoided"
222     git clone --recursive -j `nproc` https://github.com/free5gc/free5gc.
    git # clones the nightly build
223     cd free5gc
224     git -c advice.detachedHead=false checkout $FREE5GC_NIGHTLY_COMMIT #
    commit with the webconsole build and kill script fixes (among other
    updates)
225 else
```

```
226     echo "[ERROR] Script failed to set FREE5GC_STABLE_BRANCH_CONTROL
        variable"
227     exit 1
228 fi
229
230 if [ $N3IWF_STABLE_BRANCH_CONTROL -eq 0 ]; then
231     echo "[INFO] Installing N3IWF nightly"
232     echo "[INFO] Cloning N3IWF nightly branch"
233     echo "[INFO] Commit: $N3IWF_NIGHTLY_COMMIT"
234     cd NFs/n3iwf/
235     git -c advice.detachedHead=false checkout $N3IWF_NIGHTLY_COMMIT
236     cd ../../
237 fi
238
239 make # builds all the NFs
240 cd ..
241
242 ########################################
243 # Install UPF / GTP-U 5G kernel module #
244 ########################################
245 echo "[INFO] Configuring the GTP kernel module"
246 echo "[INFO] Removing GTP's previous versions, if any"
247 rm -rf gtp5g #removes previous versions
248 echo "[INFO] Installing the GTP kernel module"
249 echo "[INFO] Release: $GTP5G_VERSION"
250 git clone -c advice.detachedHead=false -b $GTP5G_VERSION https://github.
        com/free5gc/gtp5g.git
251 cd gtp5g
252 make
253 sudo make install
254 cd ..
255
256 # Install the WebConsole
257 curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
258 sudo apt update
259 sudo apt install -y nodejs
260
261 cd free5gc
262 make webconsole
263 cd ..
264
265 ################################
266 # Update the 5GC config files #
267 ################################
268 echo "[INFO] Updating configuration files"
269 ip address show $IFACENAME | grep "\binet\b"
270 # Reads the data network interface IP
```

```
271 echo "Please, type the 5GC's DN interface IP address"
272 echo -n "> "
273 read IP
274
275 # Prepare the IPSec inner tunnel IP address for N3IWF or TNGF
276 if [ $N3IWF_CONFIGURATION_CONTROL -eq 1 ] || [
    $TNGF_CONFIGURATION_CONTROL -eq 1 ]; then
277     # Get the first octet of the free5GC machine IP
278     IP_FIRST_OCTET=${IP%%.*}
279     echo "[DEBUG] free5GC machine DN interface IP 1st octet:
    $IP_FIRST_OCTET"
280
281     IP_IPSEC_INNER="10.0.0.1" # default IP is 10.0.0.1 (Check it here:
    https://github.com/free5gc/free5gc/blob/main/config/n3iwfcfg.yaml#L36
     or https://github.com/free5gc/free5gc/blob/main/config/tngfcfg.yaml#
    L36)
282     IP_IPSEC_INNER_NET_ADDR="10.0.0.0/24"
283
284     # If the UE IP belongs to the 10.x.x.x range, it will conflict with
    the IPSec tunnel address that will be added as the default route
285     if [ ${IP_FIRST_OCTET} -eq 10 ]; then
286         echo "[WARN] A conflicting IP address range for Nwu interface
    was detected"
287         echo "[INFO] Using 172.16.x.x as IPSec tunnel address space
    instead of 10.x.x.x"
288
289         IP_IPSEC_INNER="172.16.0.1" # update the IP address
290
291         IP_NET_OCTETS=`echo "$IP_IPSEC_INNER" | cut -d . -f 1-3`
292         IP_IPSEC_INNER_NET_ADDR="$IP_NET_OCTETS"".0/24" # update the
    network address
293
294         echo "[DEBUG] New IPSec tunnel inner IP address: $IP_IPSEC_INNER
    "
295         echo "[DEBUG] New IPSec tunnel IP addresses pool:
    $IP_IPSEC_INNER_NET_ADDR"
296     else
297         echo "[DEBUG] No conflicting IP address found"
298     fi
299 fi
300
301 CONFIG_FOLDER="./free5gc/config/"
302
303 # The vars below aim to find the correct line to replace the IP address.
     The commands get the line right above the one where the IP must be
    changed
304 AMF_LINE=$(grep -n 'ngapIpList:  # the IP list of N2 interfaces on this
```

```
    AMF' ${CONFIG_FOLDER}amfcfg.yaml | awk -F: '{print $1}' -)
305 SMF_LINE=$(grep -n 'endpoints: # the IP address of this N3/N9 interface
    on this UPF' ${CONFIG_FOLDER}smfcfg.yaml | awk -F: '{print $1}' -)
306 UPF_LINE=$(grep -n 'ifList:' ${CONFIG_FOLDER}upfcfg.yaml | awk -F: '{
    print $1}' -)
307 # Increment the counters to point to the next line (where the IP is
    located)
308 AMF_LINE=$((AMF_LINE+1))
309 SMF_LINE=$((SMF_LINE+1))
310 UPF_LINE=$((UPF_LINE+1))
311
312 # Update the IP on the config files
313 sed -i ""$AMF_LINE"s/.*/    - $IP/" ${CONFIG_FOLDER}amfcfg.yaml
314 sed -i ""$SMF_LINE"s/.*/              - $IP/" ${CONFIG_FOLDER}smfcfg.
    yaml
315 sed -i ""$UPF_LINE"s/.*/    - addr: $IP/" ${CONFIG_FOLDER}upfcfg.yaml
316
317 # N3IWF config
318 if [ $N3IWF_CONFIGURATION_CONTROL -eq 1 ]; then
319     N3IWF_LINE=$(grep -n '# --- N2 Interfaces ---' ${CONFIG_FOLDER}
    n3iwfcfg.yaml | awk -F: '{print $1}' -)
320     N3IWF_LINE=$((N3IWF_LINE+3))
321     sed -i ""$N3IWF_LINE"s/.*/        -  $IP/" ${CONFIG_FOLDER}n3iwfcfg.
    yaml
322     N3IWF_LINE=$((N3IWF_LINE+5))
323     if [[ $FREE5GC_VERSION = "v3.4.4" ]]; then # if new N3IWF version is
     being used
324         sed -i ""$N3IWF_LINE"s/.*/  ikeBindAddress: $IP # Nwu interface
     IP address (IKE) on this N3IWF/" ${CONFIG_FOLDER}n3iwfcfg.yaml
325         N3IWF_LINE=$((N3IWF_LINE+1))
326         sed -i ""$N3IWF_LINE"s/.*/  ipSecTunnelAddress: $IP_IPSEC_INNER
    # Tunnel IP address of XFRM interface on this N3IWF/" ${CONFIG_FOLDER
    }n3iwfcfg.yaml
327         N3IWF_LINE=$((N3IWF_LINE+1))
328         # using @ as the delimiter on the line below as
    $IP_IPSEC_INNER_NET_ADDR contains a slash that will break sed
    functionality
329         sed -i ""$N3IWF_LINE"s@.*@  ueIpAddressRange:
    $IP_IPSEC_INNER_NET_ADDR # IP address pool allocated to UE in IPSec
    tunnel@" ${CONFIG_FOLDER}n3iwfcfg.yaml
330     else # or continue using old writing style
331         sed -i ""$N3IWF_LINE"s/.*/  IKEBindAddress: $IP # Nwu interface
     IP address (IKE) on this N3IWF/" ${CONFIG_FOLDER}n3iwfcfg.yaml
332         N3IWF_LINE=$((N3IWF_LINE+1))
333         sed -i ""$N3IWF_LINE"s/.*/  IPSecTunnelAddress: $IP_IPSEC_INNER
    # Tunnel IP address of XFRM interface on this N3IWF/" ${CONFIG_FOLDER
    }n3iwfcfg.yaml
```

```
334         N3IWF_LINE=$((N3IWF_LINE+1))
335         # using @ as the delimiter on the line below as
    $IP_IPSEC_INNER_NET_ADDR contains a slash that will break sed
    functionality
336         sed -i ""$N3IWF_LINE"s@.*@  UEIPAddressRange:
    $IP_IPSEC_INNER_NET_ADDR # IP address pool allocated to UE in IPSec
    tunnel@" ${CONFIG_FOLDER}n3iwfcfg.yaml
337     echo "[INFO] N3IWF configuration applied"
338     fi
339 fi
340
341 # TNGF config
342 if [ $TNGF_CONFIGURATION_CONTROL -eq 1 ]; then
343     TNGF_LINE=$(grep -n 'AMFSCTPAddresses:' ${CONFIG_FOLDER}tngfcfg.yaml
        | awk -F: '{print $1}' -)
344     TNGF_LINE=$((TNGF_LINE+2))
345     sed -i ""$TNGF_LINE"s/.*/        - $IP/" ${CONFIG_FOLDER}tngfcfg.
    yaml
346     # TNGF_LINE=$(grep -n '# --- Bind Interfaces ---' ${CONFIG_FOLDER}
    tngfcfg.yaml | awk -F: '{print $1}' -)
347     TNGF_LINE=$((TNGF_LINE+5))
348     sed -i ""$TNGF_LINE"s/.*/  IKEBindAddress: $IP  # IP address of Nwu
    interface (IKE) on this TNGF/" ${CONFIG_FOLDER}tngfcfg.yaml
349     TNGF_LINE=$((TNGF_LINE+1))
350     sed -i ""$TNGF_LINE"s/.*/  RadiusBindAddress: $IP # IP address of
    Nwu interface (IKE) on this TNGF/" ${CONFIG_FOLDER}tngfcfg.yaml
351     TNGF_LINE=$((TNGF_LINE+1))
352     sed -i ""$TNGF_LINE"s/.*/  IPSecInterfaceAddress: $IP_IPSEC_INNER #
    IP address of IPSec virtual interface (IPsec tunnel enpoint on this
    TNGF)/" ${CONFIG_FOLDER}tngfcfg.yaml
353     TNGF_LINE=$((TNGF_LINE+1))
354     sed -i ""$TNGF_LINE"s/.*/  IPSecTunnelAddress: $IP_IPSEC_INNER #
    Tunnel IP address of XFRM interface on this TNGF/" ${CONFIG_FOLDER}
    tngfcfg.yaml
355     TNGF_LINE=$((TNGF_LINE+1))
356     # using @ as the delimiter on the line below as
    $IP_IPSEC_INNER_NET_ADDR contains a slash that will break sed
    functionality
357     sed -i ""$TNGF_LINE"s@.*@  UEIPAddressRange:
    $IP_IPSEC_INNER_NET_ADDR # IP address allocated to UE in IPSec
    tunnel@" ${CONFIG_FOLDER}tngfcfg.yaml
358     echo "[INFO] TNGF configuration applied"
359 fi
360
361 echo "[INFO] Reboot the machine to apply the new hostname"
362 if [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 1 ]; then
363     echo "[INFO] Don't forget to configure UERANSIM using the stable
```

```
    flag"
elif [ $FREE5GC_STABLE_BRANCH_CONTROL -eq 0 ]; then
    echo "[INFO] Don't forget to configure UERANSIM using the nightly
    flag"
fi
echo "[INFO] Auto deploy script done"
```

Source: Created by the author (2025).

APPENDIX C – Source of the deploy-UERANSIM.sh script

As indicated in Section 4.4, this script is part of the FAD tool. The ready-to-use source code and documentation are available in a public GitHub repository (29).

```
#!/usr/bin/env bash

echo "Welcome to the UERANSIM auto deploy script"

sudo -v # caches credentials
if [ $? == 1 ]
then
    echo "[ERROR] Without root permission, you cannot install the tools and the updates"
    exit 1
fi
echo "[INFO] Execution started"

# Control variables (1 = true, 0 = false)
CONTROL_HOSTNAME=1 # switch between updating of not the hostname
CONTROL_STABLE=0 # switch between using the free5GC stable branch or latest nightly
UERANSIM_VERSION=v3.2.6 # select the stable branch tag that will be used by the script
UERANSIM_NIGHTLY_COMMIT='' # to be used to select which commit hash will be used by the script

# check the number of parameters
if [ $# -gt 2 ]; then
    echo "[ERROR] Too many parameters given! Check your input and try again"
    exit 2
fi
if [ $# -lt 1 ]; then
    echo "[ERROR] No parameter was given! Check your input and try again"
    exit 2
fi
# check the parameters and set the control vars accordingly
if [ $# -ne 0 ]; then
    while [ $# -gt 0 ]; do
        case $1 in
            -stable)
                CONTROL_STABLE=1
                echo "[INFO] The stable branch will be cloned"
                ;;
            -nightly33)
```

```
37                  CONTROL_STABLE=0
38                  UERANSIM_NIGHTLY_COMMIT=392b714 # last commit before new SUPI/IMSI fix one (useful to be used with free5GC v3.3.0)
39                  echo "[INFO] The nightly branch to be used with free5GC v3.3.0 or below will be cloned"
40                  ;;
41              -nightly)
42                  CONTROL_STABLE=0
43                  # UERANSIM_NIGHTLY_COMMIT=e4c492d # commit with the new SUPI/IMSI fix (useful to be used with free5GC v3.4.0 or later)
44                  # UERANSIM_NIGHTLY_COMMIT=2134f6b # commit with the EAP-AKA' fix
45                  UERANSIM_NIGHTLY_COMMIT=01e3785 # commit with the Rel-17 ASN and NGAP files
46                  echo "[INFO] The nightly branch to be used with free5GC v3.4.0 or later will be cloned"
47                  ;;
48              -keep-hostname)
49                  echo "[INFO] The script will not change the machine's hostname"
50                  CONTROL_HOSTNAME=0
51          esac
52          shift
53      done
54  fi
55
56  # Hostname update
57  if [ $CONTROL_HOSTNAME -eq 1 ]; then
58      echo "[INFO] Updating the hostname"
59      sudo sed -i "1s/.*/ueransim/" /etc/hostname
60      HOSTS_LINE=$(grep -n '127.0.1.1' /etc/hosts | awk -F: '{print $1}')
61      sudo sed -i ""$HOSTS_LINE"s/.*/127.0.1.1 ueransim/" /etc/hosts
62  elif [ $CONTROL_HOSTNAME -eq 0 ]; then
63      echo "[INFO] Hostname update skipped this time"
64  else
65      echo "[ERROR] Script failed to set CONTROL_HOSTNAME variable"
66      exit 1
67  fi
68
69  ######################
70  # Download UERANSIM #
71  ######################
72  echo "[INFO] Downloading UERANSIM"
73  if [ $CONTROL_STABLE -eq 1 ]; then
74      echo "[INFO] Cloning UERANSIM stable branch"
75      echo "[INFO] Tag/release: $UERANSIM_VERSION"
```

```
76      git clone -c advice.detachedHead=false -b $UERANSIM_VERSION https://
    github.com/aligungr/UERANSIM # clones the stable build
77      cd UERANSIM
78  elif [ $CONTROL_STABLE -eq 0 ]; then
79      # first check if commit was correctly set
80      if [ -z "$UERANSIM_NIGHTLY_COMMIT" ]; then
81          echo "[ERROR] Script failed to set UERANSIM_NIGHTLY_COMMIT
    variable"
82          exit 1
83      fi
84      echo "[INFO] Cloning UERANSIM nightly branch"
85      echo "[INFO] Commit: $UERANSIM_NIGHTLY_COMMIT"
86      git clone https://github.com/aligungr/UERANSIM
87      cd UERANSIM
88      git -c advice.detachedHead=false checkout $UERANSIM_NIGHTLY_COMMIT #
     clones the nightly build
89  else
90      echo "[ERROR] Script failed to set CONTROL_STABLE variable"
91      exit 1
92  fi
93
94  ############################
95  # Install required tools #
96  ############################
97  echo "[INFO] Downloading and installing UERANSIM prerequisites"
98  sudo apt update && sudo apt upgrade -y
99  sudo apt install -y make g++ libsctp-dev lksctp-tools iproute2
100 sudo snap install cmake --classic
101
102 ###################
103 # Build UERANSIM #
104 ###################
105 echo "[INFO] Building UERANSIM"
106 make
107 cd ..
108
109 ################################
110 # Update UERANSIM config files #
111 ################################
112 echo "[INFO] Updating configuration files"
113 # Reads the data network interface IP
114 echo "Please, type the 5GC's DN interface IP address"
115 echo -n "> "
116 read IP_5GC
117 ip a
118 echo "Please, now enter the UERANSIM's N2/N3 interface IP address (e.g.
    IP that communicates with 5GC)"
```

```
119 echo -n "> "
120 read IP_UE
121
122 CONFIG_FOLDER="./UERANSIM/config/"
123
124 # The var below aim to find the correct line to replace the IP address
125 GNB_LINE=$(grep -n 'ngapIp:' ${CONFIG_FOLDER}free5gc-gnb.yaml | awk -F:
    '{print $1}' -)
126 GNB_LINE_AMF=$(grep -n 'amfConfigs:' ${CONFIG_FOLDER}free5gc-gnb.yaml |
    awk -F: '{print $1}' -)
127
128 # Increment the counter to point to the next line (where the IP is
    located)
129 GNB_LINE_AMF=$((GNB_LINE_AMF+1))
130
131 sed -i ""$GNB_LINE"s/.*/ngapIp: $IP_UE   # gNB's local IP address for N2
     Interface (Usually same with local IP)/" ${CONFIG_FOLDER}free5gc-gnb
    .yaml
132 GNB_LINE=$((GNB_LINE+1)) # go to the next line
133 sed -i ""$GNB_LINE"s/.*/gtpIp: $IP_UE   # gNB's local IP address for N3
    Interface (Usually same with local IP)/" ${CONFIG_FOLDER}free5gc-gnb.
    yaml
134 sed -i ""$GNB_LINE_AMF"s/.*/  - address: $IP_5GC/" ${CONFIG_FOLDER}
    free5gc-gnb.yaml
135
136 if [ $CONTROL_HOSTNAME -eq 1 ]; then
137     echo "[INFO] Reboot the machine to apply the new hostname"
138 fi
139 echo "[INFO] Don't forget to add the UE to the free5gc via WebConsole"
140 echo "[INFO] See: https://free5gc.org/guide/5-install-ueransim/#4-use-
    webconsole-to-add-an-ue"
141 echo "[INFO] Auto deploy script done"
```

Source: Created by the author (2025).

APPENDIX D – Source of the deploy-n3iwue.sh script

As indicated in Section 4.4, this script is part of the FAD tool. The ready-to-use source code and documentation are available in a public GitHub repository (29).

```
#!/usr/bin/env bash

echo "Welcome to the N3IWUE auto deploy script"
sudo -v # caches credentials
if [ $? == 1 ]
then
    echo "[ERROR] Without root permission, you cannot install the tools and the updates"
    exit 1
fi
echo "[INFO] Execution started"

# Control variables (1 = true, 0 = false)
HOSTNAME_CONTROL=1 # switch between updating of not the hostname
N3IWUE_VERSION=v1.0.1 # select the stable branch tag that will be used by the script
N3IWUE_STABLE_BRANCH_CONTROL=1 # switch between using the N3IWUE stable or nightly branch
N3IWUE_NIGHTLY_COMMIT='' # to be used to select which commit hash will be used by the script

# check the number of parameters
if [ $# -gt 2 ]; then
    echo "[ERROR] Too many parameters given! Check your input and try again"
    exit 2
fi
if [ $# -lt 1 ]; then
    echo "[ERROR] No parameter was given! Check your input and try again"
    exit 2
fi
# check the parameters and set the control vars accordingly
if [ $# -ne 0 ]; then
    while [ $# -gt 0 ]; do
        case $1 in
            -stable)
                echo "[INFO] The stable branch of N3IWUE will be cloned"
                ;;
            -stable341)
                N3IWUE_VERSION=v1.0.0
                echo "[INFO] The script will clone N3IWUE's version
```

```
    compatible with free5GC v3.4.1"
37                  ;;
38              -nightly)
39                  N3IWUE_STABLE_BRANCH_CONTROL=0
40                  # N3IWUE_NIGHTLY_COMMIT=c2662c7 # commit with signaling
    fixes (for more info: https://github.com/free5gc/free5gc/issues/584)
41                  N3IWUE_NIGHTLY_COMMIT=578edc9 # latest commit as of (30
    th sep)
42                  echo "[INFO] The nightly branch of N3IWUE will be cloned
    "
43                  ;;
44              -keep-hostname)
45                  HOSTNAME_CONTROL=0
46                  echo "[INFO] The script will not change the machine's
    hostname"
47                  ;;
48          esac
49          shift
50      done
51 fi
52
53 # check your go installation
54 go version
55 echo "[INFO] Go should have been previously installed, if not abort the
    execution"
56 echo "[INFO] The message above must not show a \"command not found\"
    error"
57 read -p "Press ENTER to continue or Ctrl+C to abort now"
58
59 # Hostname update
60 if [ $HOSTNAME_CONTROL -eq 1 ]; then
61     echo "[INFO] Updating the hostname"
62     sudo sed -i "1s/.*/n3iwue/" /etc/hostname
63     HOSTS_LINE=$(grep -n '127.0.1.1' /etc/hosts | awk -F: '{print $1}'
    -)
64     sudo sed -i ""$HOSTS_LINE"s/.*/127.0.1.1 n3iwue/" /etc/hosts
65 elif [ $HOSTNAME_CONTROL -eq 0 ]; then
66     echo "[INFO] Hostname update skipped this time"
67 else
68     echo "[ERROR] Script failed to set HOSTNAME_CONTROL variable"
69     exit 1
70 fi
71 ####################
72 # Download N3IWUE #
73 ####################
74 echo "[INFO] Downloading N3IWUE"
75 if [ $N3IWUE_STABLE_BRANCH_CONTROL -eq 1 ]; then
```

```
76      echo "[INFO] Cloning N3IWUE stable branch"
77      echo "[INFO] Tag/release: $N3IWUE_VERSION"
78      git clone -c advice.detachedHead=false -b $N3IWUE_VERSION https://
        github.com/free5gc/n3iwue.git # clones the stable version
79      cd n3iwue
80  elif [ $N3IWUE_STABLE_BRANCH_CONTROL -eq 0 ]; then
81      echo "[INFO] Cloning N3IWUE nightly branch"
82      echo "[INFO] Commit: $N3IWUE_VERSION"
83      git clone https://github.com/free5gc/n3iwue.git # clones the nightly
        build
84      cd n3iwue
85      git -c advice.detachedHead=false checkout $N3IWUE_NIGHTLY_COMMIT
86  else
87      echo "[ERROR] Script failed to set N3IWUE_STABLE_BRANCH_CONTROL
        variable"
88      exit 1
89  fi
90  ############################
91  # Install required tools #
92  ############################
93  echo "[INFO] Downloading and installing N3IWUE prerequisites"
94  sudo apt update && sudo apt upgrade -y
95  sudo apt install -y make libsctp-dev lksctp-tools iproute2
96  ##################
97  # Build N3IWUE #
98  ##################
99  echo "[INFO] Building N3IWUE"
100 make
101 cd ..
102 ################################
103 # Update N3IWUE config files #
104 ################################
105 echo "[INFO] Reading information required to update configuration files"
106 # Reads the 5GC data network interface IP
107 echo "Please, type the 5GC's DN interface IP address"
108 echo -n "> "
109 read IP_5GC
110 ip a
111 echo ""
112 echo "Please, enter the N3IWUE's Nwu interface name (e.g. the interface
        that communicates with 5GC)"
113 echo -n "> "
114 read IFACENAME
115 ip address show $IFACENAME | grep "\binet\b"
116 echo ""
117 # Reads the Nwu interface IP
118 echo "Please, now enter the N3IWUE's Nwu interface IP address (e.g. IP
```

```
    that communicates with 5GC)"
119 echo -n "> "
120 read IP_UE
121 # Reads the N3IWUE DN interface IP
122 echo "Please, now enter the N3IWUE's DN interface IP address (e.g. IP
    that N3WIUE will get from the 5GC)"
123 echo "TIP: If deploying only one N3IWUE to connect to the 5GC and unsure
    , then the IP should be 10.60.0.1"
124 echo -n "> "
125 read IP_DN_UE
126
127 # Get the first octet of the UE machine IP
128 IP_FIRST_OCTET=${IP_UE%%.*}
129 echo "[DEBUG] UE machine interface IP 1st octet: $IP_FIRST_OCTET"
130
131 IP_IPSEC_INNER="10.0.0.1" # default IP is 10.0.0.1 (Check it here: https
    ://github.com/free5gc/n3iwue/blob/main/config/n3ue.yaml)
132
133 # If the UE IP belongs to the 10.x.x.x range, it will conflict with the
    IPSec tunnel address that will be added as the default route
134 if [ ${IP_FIRST_OCTET} -eq 10 ]; then
135     echo "[WARN] A conflicting IP address range for Nwu interface was
    detected"
136     echo "[INFO] Using 172.16.x.x as IPSec tunnel address space instead
    of 10.x.x.x"
137
138     # To use the same host part from UE Nwu interface IP the on the
    IPSec tunnel, uncomment the lines below
139     # IP_THIRD_FOURTH_OCTETS=`echo "$IP_UE" | cut -d . -f 3-4`
140     # echo "[DEBUG] UE machine interface IP 3rd and 4th octets:
    $IP_THIRD_FOURTH_OCTETS"
141
142     # Concatenates the new range with the host part of the IP address
143     # IP_IPSEC_INNER="172.16.""$IP_THIRD_FOURTH_OCTETS" # update the IP
    address
144
145     IP_IPSEC_INNER="172.16.0.1" # update the IP address
146
147     echo "[DEBUG] New IPSec tunnel inner IP address: $IP_IPSEC_INNER"
148 fi
149
150 echo "[INFO] Updating configuration files"
151
152 CONFIG_FOLDER="./n3iwue/config/"
153 BASE_FOLDER="./n3iwue/"
154
155 # The var below aim to find the correct line to replace the IP address
```

```
156 N3IWF_LINE=$(grep -n 'N3IWFInformation:' ${CONFIG_FOLDER}n3ue.yaml | awk
      -F: '{print $1}' -)
157 N3UE_LINE=$(grep -n 'IPSecIfaceName: ens38 # Name of Nwu interface (IKE)
      on this N3UE' ${CONFIG_FOLDER}n3ue.yaml | awk -F: '{print $1}' -)
158 N3UE_RUN_SCRIPT_IPSEC_LINE=$(grep -n 'N3UE_IPSec_iface_addr=' ${
    BASE_FOLDER}run.sh | awk -F: '{print $1}' -)
159
160 # Increment the counter to point to the next line (where the IP is
    located)
161 N3IWF_LINE=$((N3IWF_LINE+1))
162
163 # Update the IP on the config files
164 sed -i ""$N3IWF_LINE"s/.*/        IPSecIfaceAddr: $IP_5GC # IP address
    of Nwu interface (IKE) on N3IWF/" ${CONFIG_FOLDER}n3ue.yaml
165 N3IWF_LINE=$((N3IWF_LINE+1)) # go to the next line
166 sed -i ""$N3IWF_LINE"s/.*/        IPsecInnerAddr: $IP_IPSEC_INNER # IP
    address of IPsec tunnel enpoint on N3IWF/" ${CONFIG_FOLDER}n3ue.yaml
167
168 sed -i ""$N3UE_LINE"s/.*/        IPSecIfaceName: $IFACENAME # Name of
    Nwu interface (IKE) on this N3UE/" ${CONFIG_FOLDER}n3ue.yaml
169 N3UE_LINE=$((N3UE_LINE+1)) # go to the next line
170 sed -i ""$N3UE_LINE"s/.*/        IPSecIfaceAddr: $IP_UE # IP address of
    Nwu interface (IKE) on this N3UE/" ${CONFIG_FOLDER}n3ue.yaml
171
172 # Update the IP on the run script file
173 sed -i ""$N3UE_RUN_SCRIPT_IPSEC_LINE"s/.*/    N3UE_IPSec_iface_addr=
    $IP_5GC/" ${BASE_FOLDER}run.sh
174 N3UE_RUN_SCRIPT_IPSEC_LINE=$((N3UE_RUN_SCRIPT_IPSEC_LINE+1)) # go to the
     next line
175 sed -i ""$N3UE_RUN_SCRIPT_IPSEC_LINE"s/.*/    N3IWF_IPsec_inner_addr=
    $IP_IPSEC_INNER/" ${BASE_FOLDER}run.sh
176 N3UE_RUN_SCRIPT_IPSEC_LINE=$((N3UE_RUN_SCRIPT_IPSEC_LINE+1)) # go to the
     next line
177 sed -i ""$N3UE_RUN_SCRIPT_IPSEC_LINE"s/.*/    UE_DN_addr=$IP_DN_UE/" ${
    BASE_FOLDER}run.sh
178
179 if [ $HOSTNAME_CONTROL -eq 1 ]; then
180     echo "[INFO] Reboot the machine to apply the new hostname"
181 fi
182 echo "[INFO] Don't forget to add the N3IWUE to free5GC via WebConsole"
183 echo "[INFO] See: https://free5gc.org/guide/n3iwue-installation/#3-use-
    webconsole-to-add-ue"
184 echo "[INFO] Auto deploy script done"
```

Source: Created by the author (2025).

APPENDIX E – Source of the deploy-tngfue.sh script

As indicated in Section 4.4, this script is part of the FAD tool. The ready-to-use source code and documentation are available in a public GitHub repository (29).

```
#!/usr/bin/env bash
WLAN_IFACE_NAME="wlp3s0"

echo "Welcome to the TNGFUE auto deploy script"
sudo -v # caches credentials
if [ $? == 1 ]
then
    echo "[ERROR] Without root permission, you cannot install the tools and the updates"
    exit 1
fi
echo "[INFO] Execution started"

# Control variables (1 = true, 0 = false)
CONTROL_HOSTNAME=1 # switch between updating or not the hostname
CONTROL_STABLE=0 # switch between using the TNGFUE stable branch or latest nightly
# TNGFUE_VERSION=TODO # select the stable branch tag that will be used by the script
TNGFUE_NIGHTLY_COMMIT='' # to be used to select which commit hash will be used by the script

# check the number of parameters
if [ $# -gt 2 ]; then
    echo "[ERROR] Too many parameters given! Check your input and try again"
    exit 2
elif [ $# -lt 1 ]; then
    echo "[ERROR] No parameter was given! Check your input and try again"
    exit 2
fi
# check the parameters and set the control vars accordingly
if [ $# -ne 0 ]; then
    while [ $# -gt 0 ]; do
        case $1 in
            -stable)
                CONTROL_STABLE=1
                # echo "[INFO] The stable branch will be cloned"
                echo "[ERROR] No stable branch/tag available!"
                ;;
            -nightly)
```

```
37                  CONTROL_STABLE=0
38                  TNGFUE_NIGHTLY_COMMIT=63823f7 # commit with the new
    execution flow (automated scripts)
39                  echo "[INFO] The nightly branch will be cloned"
40                  ;;
41              -keep-hostname)
42                  echo "[INFO] The script will not change the machine's
    hostname"
43                  CONTROL_HOSTNAME=0
44          esac
45          shift
46      done
47 fi
48
49 # Hostname update
50 if [ $CONTROL_HOSTNAME -eq 1 ]; then
51      echo "[INFO] Updating the hostname"
52      sudo sed -i "1s/.*/ueransim/" /etc/hostname
53      HOSTS_LINE=$(grep -n '127.0.1.1' /etc/hosts | awk -F: '{print $1}'
    -)
54      sudo sed -i ""$HOSTS_LINE"s/.*/127.0.1.1 ueransim/" /etc/hosts
55 elif [ $CONTROL_HOSTNAME -eq 0 ]; then
56      echo "[INFO] Hostname update skipped this time"
57 else
58      echo "[ERROR] Script failed to set CONTROL_HOSTNAME variable"
59      exit 1
60 fi
61 ########################
62 # TNGFUE Installation #
63 ########################
64 echo "[INFO] Downloading TNGFUE"
65 git clone https://github.com/free5gc/tngfue.git
66 cd tngfue
67 git -c advice.detachedHead=false checkout $TNGFUE_NIGHTLY_COMMIT #
    clones the nightly build
68 echo "[INFO] Running the prepare.sh script"
69 ./prepare.sh
70
71 if [ $CONTROL_HOSTNAME -eq 1 ]; then
72      echo "[INFO] Reboot the machine to apply the new hostname"
73 fi
74 echo "[INFO] Don't forget to add the TNGFUE to free5GC via WebConsole"
75 echo "[INFO] See: https://free5gc.org/guide/TNGF/tngfue-installation/#2-
    use-webconsole-to-add-ue"
76 echo "[INFO] Auto deploy script done"
```

Source: Created by the author (2025).

APPENDIX F – Source of the pcap_extract.sh script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
#!/usr/bin/env bash

TRAINING_PCAP_FOLDER=./pcap/input/training_data/ # read training raw PCAP files from here
INFERENCE_PCAP_FOLDER=./pcap/input/inference_data/ # read inference raw PCAP files from here
OUT_FOLDER=./pcap/output/1-PCAP-export/ # save the output there

# set IFS to break only on new line (so the input files can have white space on their names)
# for more info: https://www.linuxquestions.org/questions/programming-9/bash-put-output-from-%60ls%60-into-an-array-346719/#post1765355
IFS='
'

TRAINING_PCAP_LIST=("$TRAINING_PCAP_FOLDER"*.pcap)
INFERENCE_PCAP_LIST=("$INFERENCE_PCAP_FOLDER"*.pcap)

# Check if folder is empty, if yes, delete the related var to avoid "file not found" errors
if [[ "$TRAINING_PCAP_LIST" == *"*"* ]]; then
    # TRAINING_PCAP_LIST=""
    unset TRAINING_PCAP_LIST
elif [[ "$INFERENCE_PCAP_LIST" == *"*"* ]]; then
    # INFERENCE_PCAP_LIST=""
    unset INFERENCE_PCAP_LIST
fi

# Then calculate these sizes after that
TRAINING_PCAP_LIST_SIZE=${#TRAINING_PCAP_LIST[@]}
INFERENCE_PCAP_LIST_SIZE=${#INFERENCE_PCAP_LIST[@]}
TOTAL_LIST_SIZE=$((TRAINING_PCAP_LIST_SIZE + INFERENCE_PCAP_LIST_SIZE))

# TIME_START=$(date +%s) # record start time

# PCAP to JSON and CSV
extract_JSON_and_CSV () {
    local FILE_NAME=$1
    local PCAP_FOLDER=$2
    local OUT_FOLDER=$3
    local SET_SPLIT_TAG=$4

```

```
38      echo "[INFO] Extracting data from $FILE_NAME"
39
40      tshark -r $PCAP_FOLDER$FILE_NAME -T json > $OUT_FOLDER/"${FILE_NAME
    %.*}_$SET_SPLIT_TAG.json" && \
41      tshark -r $PCAP_FOLDER$FILE_NAME -T fields \
42      -e frame.number -e frame.time_relative -e ip.src -e ip.dst -e _ws.
    col.protocol -e frame.len -e _ws.col.info \
43      -E header=y -E separator=, -E quote=d -E occurrence=f \
44      > $OUT_FOLDER/"${FILE_NAME%.*}_$SET_SPLIT_TAG.csv"
45      # To customize the "-e flags" (display filters), see https://www.
    wireshark.org/docs/dfref/
46      # For more information: https://manpages.ubuntu.com/manpages/jammy/
    man1/tshark.1.html
47  }
48
49  # JSON field remover
50  field_remover () {
51      local FILE_NAME=$1
52      local OUT_FOLDER=$2
53
54      echo "[INFO] Removing unused fields from $FILE_NAME"
55      # GNU AWK (gawk) was used instead of AWK because of this regex
    support
56      gawk -i inplace '/\[/ || /\]/ || /\{/ || /\}/ || /"_source":/ ||
57      /"frame.number":/ || /"frame.time_delta":/ || /"frame.time_relative
    ":/ ||
58      /"ip.src":/ || /"ip.dst":/ || /"frame.encap_type":/ || /"frame.len
    ":/ ||
59      /"ip.hdr_len":/ || /"udp.length":/ || /"tcp.len":/ || /"udp.srcport
    ":/ ||
60      /"tcp.srcport":/ || /"udp.dstport":/ || /"tcp.dstport":/ ||
61      /"tcp.completeness*"/ || /"tcp.flags":/ || /"tcp.str":/ ||
62      /"tcp.window_size":/ || /"tcp.window_size_scalefactor":/ ||
63      /"frame.protocols":/ || /"ip.proto":/ || /"ip.flags...":/ || /"ip.
    ttl":/ ||
64      /"tcp.hdr_len":/ || /"data.len":/ || /"quic.packet_length":/ || /"
    quic.length":/
65      ' $OUT_FOLDER$FILE_NAME
66      # Some regex pattern explanation
67      # first line of the pattern matches the JSON structure header that
    must be kept
68      # all the other pattern lines match lines that contain data to be
    used on next steps
69      # everything else that is not matched, will be deleted to free disk
    space
70  }
71
```

```
72 # JSON parser to reconstruct its structure
73 json_parser () {
74     local FILE_NAME=$1
75     local OUT_FOLDER=$2
76     FILE_NAME_TMP=$FILE_NAME"_tmp"
77 
78     echo "[INFO] Parsing fields from $FILE_NAME"
79     # Parses the JSON using the Perl module to remove illegal trailing
       commas and to reformat it
80     perl -MJSON -e '@text=(<>);print to_json(from_json("@text", {relaxed
       =>1}), {pretty=>1})' $OUT_FOLDER$FILE_NAME >
       $OUT_FOLDER$FILE_NAME_TMP
81     # the Perl source code above was based on this StackOverflow answer:
       https://unix.stackexchange.com/a/485011
82     # As I couldn't find an easy way to process the file in place, I've
       added a tem file to save the result then used mv to overwrite the
       original
83     mv $OUT_FOLDER$FILE_NAME_TMP $OUT_FOLDER$FILE_NAME
84 }
85 
86 time { # track execution time
87 echo "[INFO] Exporting $TRAINING_PCAP_LIST_SIZE training and
       $INFERENCE_PCAP_LIST_SIZE inference PCAP files"
88 if [ $TRAINING_PCAP_LIST_SIZE -ne 0 ]; then
89     for i in "${TRAINING_PCAP_LIST[@]}"; do
90         extract_JSON_and_CSV "${i##*/}" $TRAINING_PCAP_FOLDER
       $OUT_FOLDER "training" &
91     done
92 fi
93 if [ $INFERENCE_PCAP_LIST_SIZE -ne 0 ]; then
94     for j in "${INFERENCE_PCAP_LIST[@]}"; do
95         extract_JSON_and_CSV "${j##*/}" $INFERENCE_PCAP_FOLDER
       $OUT_FOLDER "inference" &
96     done
97 fi
98 wait
99 unset TRAINING_PCAP_LIST # clean up after usage
100 unset INFERENCE_PCAP_LIST_SIZE # clean up after usage
101 echo "[INFO] All $TOTAL_LIST_SIZE files have been successfully exported"
102 
103 JSON_LIST=("$OUT_FOLDER"*.json)
104 JSON_LIST_SIZE=${#JSON_LIST[@]}
105 
106 # Drop duplicated fields on JSON
107 echo "[INFO] Preprocessing entries from $JSON_LIST_SIZE JSON files"
108 if [ $JSON_LIST_SIZE -ne 0 ]; then
109     for i in "${JSON_LIST[@]}"; do
```

```
110         field_remover "${i##*/}" $OUT_FOLDER &
111     done
112     wait
113     for i in "${JSON_LIST[@]}"; do
114         # this step uses a lot of RAM, that's why it ins't run in
    parallel
115         json_parser "${i##*/}" $OUT_FOLDER
116     done
117 else
118     echo "[ERRO] No JSON files found in the directory: $OUT_FOLDER"
119     exit 1
120 fi
121 unset JSON_LIST # clean up after usage
122 echo "[INFO] All $TOTAL_LIST_SIZE files have been processed"
123 
124 echo "[DEBU] Execution time:"
125 }
126 
127 # TIME_END=$(date +%s) # record end time
128 # echo "[DEBU] Execution time: $((TIME_END-TIME_START)) seconds"
```

Source: Created by the author (2025).

APPENDIX G – Source of the dataset_CSV_characterization.py script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import csv
import traceback
import sys
import pandas as pd
import os
import glob
import time

from util import read_csv

def extract_frequency_info(data_frames, column_names):
    freq_data_list = []

    for df in data_frames:
        try:
            # Use value_counts to get the frequency of each unique value in the specified columns
            frequency_info = {column: df[column].value_counts() for column in column_names}
            freq_data_list.append(frequency_info)
        except KeyError as ke:
            missing_columns = [col for col in column_names if col not in df.columns]
            print(f"Error: Columns {missing_columns} not found in the DataFrame.")
            exit()
        except Exception as e:
            # printing stack trace
            traceback.print_exception(*sys.exc_info())
            print("[ERROR]", type(e).__name__, e)
            exit()

    return freq_data_list

def print_and_save_frequency_data(freq_data_list):
    for counter, item in enumerate(freq_data_list, start=1):
        # print("[DEBU] Frequency data extracted from data frame number ", counter) # DEBUG
        for column_name, freq_series in item.items():
            file_name_without_format = os.path.splitext(input_files_names[counter - 1])[0] # remove '.csv' from old file name
            freq_series.to_csv(os.path.join(output_files_path,
```

```
    file_name_without_format + "." + column_name + ".csv"))
37              # print(f"[DEBU] Frequency information for {column_name} of
    {file_name_without_format}:\n {freq_series}") # DEBUG
38          # print("[DEBU] Finished printing data frame", counter) # DEBUG
39
40 def save_time_data(df_list):
41     # Extract the two columns related to time from each DataFrame
42     for counter, time_df in enumerate(df_list):
43         extracted_cols = pd.DataFrame()
44         extracted_cols = (time_df[["frame.number", "frame.time_relative"
    ]])
45         file_name_without_format = os.path.splitext(input_files_names[
    counter - 1])[0] # remove '.csv' from old file name
46         extracted_cols.to_csv(os.path.join(output_files_path,
    file_name_without_format + ".time_series.csv"), index=False)
47
48 # File paths
49 input_files_path = "./pcap/output/1-PCAP-export/" # read CSV files from
    here
50 output_files_path = "./pcap/output/2-stats/" # save the output there
51
52 print("[INFO] Creating statistical data")
53 start_time = time.time() # record the start of execution
54
55 # get the list of all CSV files in the input directory
56 input_files_names = [f for f in os.listdir(input_files_path) if f.
    endswith('.csv')]
57 # input_files_names = [f for f in os.listdir(input_files_path) if f.
    endswith('test.csv') | f.endswith('5g1.csv')] # initial tests
58 # create a list of file paths by joining the input directory path with
    each file name
59 input_file_paths = [os.path.join(input_files_path, f) for f in
    input_files_names]
60
61 # Read CSV files
62 input_dfs = [read_csv(path) for path in input_file_paths]
63
64 # check if at least one file was found
65 if len(input_dfs) == 0:
66     print(f"[ERROR] No CSV file found on {input_files_path}")
67     exit()
68
69 # Extract frequency information
70 # as the features are the same on all files, no need to do that for all
    of them
71 column_names = input_dfs[0].columns.tolist()
72 # removes some columns from the frequency calculation because they
```

```
    almost always have only unique values
columns_to_remove = ["frame.number", "frame.time_relative", "_ws.col.
    info"]

for col in columns_to_remove:
    column_names.remove(col)

input_freq_data_list = extract_frequency_info(input_dfs, column_names)

# Print frequency information
print_and_save_frequency_data(input_freq_data_list)
save_time_data(input_dfs)
print("[INFO] Creating statistical data successfully finished")
end_time = time.time() # record the end of execution
print(f"[DEBU] Execution time: {(end_time - start_time)} s")
```

Source: Created by the author (2025).

APPENDIX H – Source of the stat-plotter.py script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import csv
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import time

from util import read_csv

def update_font_size(f_size):
    font_size = f_size # base font size
    # Update rcParams to make fonts larger
    plt.rcParams['font.size'] = font_size
    plt.rcParams['axes.labelsize'] = font_size + 2
    plt.rcParams['axes.titlesize'] = font_size + 4
    plt.rcParams['xtick.labelsize'] = font_size
    plt.rcParams['ytick.labelsize'] = font_size

# Create chart for each DataFrame
def plot_graph(df_to_plot, input_file_name, column_label, x_label,
    y_label, plt_type):
    for i, df in enumerate(df_to_plot):

        # Adjust plot parameters according to each plot type
        if (plt_type == 'dozens-of-bars'):
            plt.figure(figsize=(15, 6)) # Set figure size
            update_font_size(15)
            sorted_df = df.sort_values(by=column_label) # sort data
    before plotting
            # plt.plot(sorted_df[column_label], sorted_df['count'],
    marker='.', linestyle=':') # line plot
            plt.bar(sorted_df[column_label], sorted_df['count']) # bar
    plot

            # Calculate mean, median and mode
            mean_value = df['frame.len'].mean()
            median_value = df['frame.len'].median()
            mode_value = df['frame.len'][0]

            # Statistical bars configuration
            bar_max_height = sorted_df['count'].max() # get the highest
    height value on the plot
```

```
38            text_x_pos = 0.05 # x axis text anchor
39            text_y_pos = 0.95 # y axis text anchor
40            # set the colors
41            mean_color = 'red'
42            median_color = 'green'
43            mode_color = 'purple'
44
45            # Plot mean, median and mode as colored bars
46            plt.bar(mean_value, bar_max_height, color=mean_color, alpha
    =0.5, width=3)
47            plt.bar(median_value, bar_max_height, color=median_color,
    alpha=0.5, width=3)
48            plt.bar(mode_value, bar_max_height, color=mode_color, alpha
    =0.5, width=3)
49
50            # Annotate the plot with mean, median and mode values
51            plt.text(text_x_pos, text_y_pos, f'Mean: {mean_value:.2f}',
    transform=plt.gca().transAxes, ha='left', va='top', color=mean_color)
52            plt.text(text_x_pos, (text_y_pos - 0.05), f'Median: {
    median_value:.2f}', transform=plt.gca().transAxes, ha='left', va='top
    ', color=median_color)
53            plt.text(text_x_pos, (text_y_pos - 0.10), f'Mode: {
    mode_value}', transform=plt.gca().transAxes, ha='left', va='top',
    color=mode_color)
54
55            # Adjust the grid and x axis labels
56            plt.xticks(np.arange(0, 1505, 50), rotation=30)
57            plt.grid(visible=True, axis='y', linestyle = '--', zorder=0)
58
59        elif (plt_type == 'a-few-bars'):
60            plt.figure(figsize=(10, 6)) # Set figure size
61            update_font_size(12)
62            x = df[column_label]
63            y = df['count'] # get the count column data
64            labels = [str(x) for x in df[column_label]] # convert all
    labels to strings (required by plt.barh())
65            total_count = y.sum()
66
67            # Map each xlabel to a specific color
68            protocol_labels_list = ["UDP",
69                                    "ICMPv6",
70                                    "TCP", "TCP, HiPerConTracer",
71                                    "TLSv1.3", "TLSv1.2", "TLSv1",
72                                    "SSLv2", "SSL",
73                                    "H1",
74                                    "HTTP", "HTTP/JSON",
75                                    "DNS",
```

```
76                                      "QUIC",
77                                      "PNIO",
78                                      "OCSP"]
79              color_labels = ['blue',
80                              'purple',
81                              'green', 'green',
82                              'orange', 'orange', 'orange',
83                              'black', 'black',
84                              'magenta',
85                              'brown', 'brown',
86                              'pink',
87                              'grey',
88                              'teal',
89                              'cyan']
90              label_to_color = {label: color for label, color in zip(
    protocol_labels_list, color_labels)}
91              bar_colors = [label_to_color[label] for label in labels]
92
93              plt.bar(x, y, align='center', color=bar_colors, label=labels
    )
94              plt.xticks(x, labels, rotation=15)
95
96              # Offsets tailored for the used dataset
97              # they were obtained via trial and error
98              if (y.max() > 1000):
99                  lim_upper_offset = 4.0
100             else:
101                 lim_upper_offset = 1.5
102             lim_bottom_offset = 1.4
103
104             plt.ylim(y.min() / lim_bottom_offset, lim_upper_offset * y.
    max()) # adjust the bars to avoid plotting text out of bounds
105             # Add the counts and percentages as labels above each bar
106             for j in range(len(y)):
107                 percentage = round((y[j]/total_count)*100, 1)
108                 label_text = f"{y[j]}\n({percentage}%)" # format the
    label text with both count and percentage
109                 plt.text(j, y[j], label_text, ha='center', va='bottom')
110         else:
111             print("[ERROR] Could not set plt_type correctly, currently
    it is:", plt_type)
112             exit()
113
114         plt.yscale('log')
115         file_name_without_format = os.path.splitext(input_file_name[i])
    [0] # remove '.csv' from old file name
116         plt.title(file_name_without_format)
```

```
117         plt.xlabel(x_label)
118         plt.ylabel(y_label + " (Logarithmic Scale)")
119         plt.tight_layout()
120 
121         # Save plot
122         output_file_path = os.path.join(output_files_path, f"{
    file_name_without_format}.pdf")
123         plt.savefig(output_file_path, dpi=600, bbox_inches="tight")
124         print(f"[INFO] Plots of {x_label} for {input_file_name[i]} have
    been saved") # TODO improve messages on screen
125 
126         # plt.show() # DEBUG
127         plt.clf()  # clear the figure to create a new plot
128         plt.close() # close each figure after finishing to free RAM
129 
130 def plot_time_series(dfs_to_plot, input_file_name, x_column_label,
    y_column_label, x_label, y_label):
131     for i, df in enumerate(dfs_to_plot):
132         plt.figure(figsize=(15, 6)) # Set figure size
133         plt.plot(df[x_column_label], df[y_column_label], marker='.',
    linestyle=':')
134         file_name_without_format = os.path.splitext(input_file_name[i])
    [0] # remove '.csv' from old file name
135         plt.title(file_name_without_format)
136         plt.xlabel(x_label + " (seconds)")
137         plt.ylabel(y_label)
138         plt.tight_layout()
139 
140         # Save plot
141         output_file_path = os.path.join(output_files_path, f"{
    file_name_without_format}.pdf")
142         plt.savefig(output_file_path, dpi=120, bbox_inches="tight")
143         print(f"[INFO] Plots of {x_label} for {input_file_name[i]} have
    been saved") # TODO improve messages on screen
144 
145         # plt.show() # DEBUG
146         plt.clf()  # clear the figure to create a new plot
147         plt.close() # close each figure after finishing to free RAM
148 
149 # File paths
150 input_files_path = "./pcap/output/2-stats/" # read CSV files from here
151 output_files_path = "./pcap/output/2-stats/graphs/" # save the output
    there
152 
153 start_time = time.time() # record the start of execution
154 # Get file names and paths for protocol and length data
155 input_file_names_protocol = [f for f in os.listdir(input_files_path) if
```

```
    f.endswith('protocol.csv')]
156 input_file_paths_protocol = [os.path.join(input_files_path, f) for f in
    input_file_names_protocol]
157 input_file_names_length = [f for f in os.listdir(input_files_path) if f.
    endswith('len.csv')]
158 input_file_paths_length = [os.path.join(input_files_path, f) for f in
    input_file_names_length]
159 input_file_names_frame_time_number = [f for f in os.listdir(
    input_files_path) if f.endswith('time_series.csv')]
160 input_file_paths_frame_time_number = [os.path.join(input_files_path, f)
    for f in input_file_names_frame_time_number]
161
162 # Read CSV files
163 input_dfs_protocol = [read_csv(path) for path in
    input_file_paths_protocol]
164 input_dfs_length = [read_csv(path) for path in input_file_paths_length]
165 input_dfs_time_series = [read_csv(path) for path in
    input_file_paths_frame_time_number]
166
167 # Check if at least one file was found for each type
168 # TODO improve this check to filter per type
169 if (not input_dfs_protocol and not input_dfs_length and not
    input_dfs_time_series):
170     print(f"[ERROR] No CSV files found on {input_files_path}")
171     exit()
172
173 # Create plots for both protocol and length data
174 plot_graph(input_dfs_protocol, input_file_names_protocol, '_ws.col.
    protocol', 'Protocol Label', 'Frequency', 'a-few-bars')
175 plot_graph(input_dfs_length, input_file_names_length, 'frame.len', '
    Packet Length (bytes)', 'Frequency', 'dozens-of-bars')
176 plot_time_series(input_dfs_time_series,
    input_file_names_frame_time_number, 'frame.time_relative', 'frame.
    number', 'Packet Capture Time', 'Packet Number')
177
178 print("[INFO] All plots have been finished")
179 end_time = time.time() # record the end of execution
180 print(f"[DEBU] Execution time: {end_time - start_time} s")
```

Source: Created by the author (2025).

APPENDIX I – Source of the export_JSON.py script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import os
import time

from joblib import Parallel, delayed
from pcap_json2csv.json2csv import parse

start_time = time.time() # record the start of execution
number_of_parallel_jobs=int(os.cpu_count()/2) # take half of reported CPU threads
# File paths
input_files_path = "./pcap/output/1-PCAP-export/" # read JSON files from here
output_files_path = "./pcap/output/3-JSON-export/" # save the output there

# get the list of all JSON files in the input directory
input_files_names = [f for f in os.listdir(input_files_path) if f.endswith('.json')]
# input_files_names = [f for f in os.listdir(input_files_path) if f.endswith('test.json') | f.endswith('5g1.json')] # initial tests
# create a list of file paths by joining the input directory path with each file name
input_file_paths = [os.path.join(input_files_path, f) for f in input_files_names]

# check if at least one file was found
if not input_file_paths:
    print(f"[ERRO] No JSON file found on {input_files_path}")
    exit()

# parse all JSON files read
def export_json(file, progress_bar_slow_print):
    try:
        parse(file, output_files_path, progress_bar_slow_print)
    except KeyboardInterrupt:
        print("\n[ERRO] Operation interrupted by the user")
        exit()
    print(f"\n[INFO] Export of {file} done")

try:
    slowdown_print=True
    print(f"[INFO] Running in parallel using {number_of_parallel_jobs}
```

```
    threads")
36      Parallel(n_jobs=number_of_parallel_jobs)(delayed(export_json)(i,
    slowdown_print) for i in input_file_paths)
37 except KeyboardInterrupt:
38      print("[ERRO] User asked to quit")
39      exit()
40 # swap the lines on the block above with these below to disable the
    parallel execution
41 # for i in input_file_paths:
42 #     slowdown_print=False
43 #     export_json(i, slowdown_print)
44 end_time = time.time() # record the end of execution
45 print("[DEBU] Execution time:", end_time - start_time, "s")
```

Source: Created by the author (2025).

APPENDIX J – Source of the json2csv.py script

As indicated in Subsection **4.5.1**, this script is part of the `PCAP-dataExtractor` module. The ready-to-use source code is available in a GitHub repository (122).

```
import json
import pandas as pd
import os
import sys

def swap_special_chars_to_underscore(input_string):
    result = '' # to store the output

    # Check if the character is alphanumeric
    # If alphanumeric, add the character to the result string
    # If not alphanumeric, add an underscore to the result string
    for char in input_string:
        if char.isalnum():
            result += char
        else:
            result += '_'

    return result

def progressBar(count_value, total, slow_print, suffix=''):
    """
    A simple and tidy progress bar to track status
    """
    # Adapted from: https://www.geeksforgeeks.org/progress-bars-in-python/
    bar_length = 25
    filled_up_Length = int(round(bar_length* count_value / float(total)))
    percentage = round(100.0 * count_value/float(total),1)
    bar = '=' * filled_up_Length + ' ' * (bar_length - filled_up_Length)

    # This new way of printing improves when using parallelization
    # if on slow mode and file is large, update for each 1k packets
    # if on slow mode and file is medium sized, update for each 100 packets
    # else if on slow mode and file is small, update for each 10 packets
    # else if not on slow mode, update at every packet (the original way)
    # else don't do anything (i.e. skip)
    if (slow_print and total >= 10000 and count_value % 1000 == 0):
        sys.stdout.write('[%s] %s%s: %s\n' %(bar, percentage, '%', suffix))
```

```
38      elif (slow_print and total > 1000 and total < 10000 and count_value
    % 100 == 0):
39          sys.stdout.write('[%s] %s%s: %s\n' %(bar, percentage, '%',
    suffix))
40      elif (slow_print and total <= 1000 and count_value % 10 == 0):
41          sys.stdout.write('[%s] %s%s: %s\n' %(bar, percentage, '%',
    suffix))
42      elif (not slow_print):
43          sys.stdout.write('[%s] %s%s: %s\r' %(bar, percentage, '%',
    suffix))
44
45  def parse(input_file_path, output_folder, print_control=False):
46      """
47      A function to parse JSON PCAP-style files to a CSV format
48      """
49      # check if path is empty then ask user to provide it
50      if (not input_file_path):
51          print("[ERRO] The file path was not provided!")
52          print("[INFO] Please, provide the file path below")
53          input_file_path = input("> ")
54
55      file_path_without_format = os.path.splitext(input_file_path)[0] #
    remove '.json' from old file name
56      file_name_without_format = file_path_without_format.split("/")
57      file_name_without_format = file_name_without_format[len(
    file_name_without_format)-1] # get the clean file name only
58      new_file_name = file_name_without_format + '.csv'
59      new_file_with_path = output_folder + new_file_name
60
61      JSON_data=open(input_file_path).read()
62      print(f"[INFO] JSON file {file_name_without_format} opened
    successfully")
63      JSONArray = json.loads(JSON_data)
64      print("[INFO] JSON data imported successfully")
65
66      length = len(JSONArray)
67      print("[DEBU]", length, "data frames to be converted from",
    file_name_without_format) # DEBUG
68
69      labels = ['Packet_no','Timestamp','Time_delta','Source_IP','
    Destination_IP','Frame_type','Frame_total_length','
    Frame_header_length',
70              'Frame_payload_length','Source_port','Destination_port','
    TCP_completeness','TCP_compl_reset','TCP_compl_fin','TCP_compl_data',
71              'TCP_compl_ack','TCP_compl_syn_ack','TCP_compl_syn','
    TCP_compl_str','TCP_flags_bin','TCP_flags_str','TCP_window_size',
72              'TCP_window_size_scale','Frame_protocols','IP_protocols','
```

```
    IP_flag_reserved_bit','IP_flag_dont_fragment','IP_flag_more_fragments
    ',
73              'TTL', 'TCP_header_length','Data_length','QUIC_packet_length
    ','QUIC_length']
74
75      df = pd.DataFrame(columns = labels)
76      df.to_csv(new_file_with_path, header = True, index = False)
77      for obj in range(length):
78
79          try:
80              pkt_number = JSONArray[obj]['_source']['layers']['frame']['
    frame.number']
81          except Exception:
82              timestamp = None
83
84          try:
85              timestamp = JSONArray[obj]['_source']['layers']['frame']['
    frame.time_relative']
86          except Exception:
87              timestamp = None
88
89          try:
90              time_since_last_pkt = JSONArray[obj]['_source']['layers']['
    frame']['frame.time_delta']
91          except Exception:
92              time_since_last_pkt = None
93
94          try:
95               ipv4_ip_src = JSONArray[obj]['_source']['layers']['ip']['ip
    .src']
96          except Exception:
97              ipv4_ip_src = None
98
99          try:
100             ipv4_ip_dst = JSONArray[obj]['_source']['layers']['ip']['ip.
    dst']
101         except Exception:
102             ipv4_ip_dst = None
103
104         try:
105             frame_type = JSONArray[obj]['_source']['layers']['frame']['
    frame.encap_type']
106         except Exception:
107             frame_type = None
108             # more info on that: https://gitlab.com/wireshark/wireshark
    /-/blob/master/wiretap/wtap.h#L87
109
```

```
110         try:
111             frame_len = JSONArray[obj]['_source']['layers']['frame']['
    frame.len']
112         except Exception:
113             frame_len = None
114 
115         try:
116             header_len = JSONArray[obj]['_source']['layers']['ip']['ip.
    hdr_len']
117         except Exception:
118             header_len = None
119 
120         try:
121             payload_len = JSONArray[obj]['_source']['layers']['udp']['
    udp.length']
122         except Exception:
123             try:
124                 payload_len = JSONArray[obj]['_source']['layers']['tcp'
    ]['tcp.len']
125             except Exception:
126                 payload_len = None
127 
128         try:
129             src_port = JSONArray[obj]['_source']['layers']['udp']['udp.
    srcport']
130         except Exception:
131             try:
132                 src_port = JSONArray[obj]['_source']['layers']['tcp']['
    tcp.srcport']
133             except Exception:
134                 src_port = None
135 
136         try:
137             dst_port = JSONArray[obj]['_source']['layers']['udp']['udp.
    dstport']
138         except Exception:
139             try:
140                 dst_port = JSONArray[obj]['_source']['layers']['tcp']['
    tcp.dstport']
141             except Exception:
142                 dst_port = None
143 
144         # TCP only begin #
145         try:
146             tcp_completeness = JSONArray[obj]['_source']['layers']['tcp'
    ]['tcp.completeness']
147         except Exception:
```

```
148             tcp_completeness = None
149
150         # TODO forcibly convert these "completeness flags" to int or
    bool
151         try:
152             tcp_completeness_reset = JSONArray[obj]['_source']['layers'
    ]['tcp']['tcp.completeness_tree']['tcp.completeness.rst']
153         except Exception:
154             tcp_completeness_reset = None
155
156         try:
157             tcp_completeness_fin = JSONArray[obj]['_source']['layers']['
    tcp']['tcp.completeness_tree']['tcp.completeness.fin']
158         except Exception:
159             tcp_completeness_fin = None
160
161         try:
162             tcp_completeness_data = JSONArray[obj]['_source']['layers']['
    tcp']['tcp.completeness_tree']['tcp.completeness.data']
163         except Exception:
164             tcp_completeness_data = None
165
166         try:
167             tcp_completeness_ack = JSONArray[obj]['_source']['layers']['
    tcp']['tcp.completeness_tree']['tcp.completeness.ack']
168         except Exception:
169             tcp_completeness_ack = None
170
171         try:
172             tcp_completeness_syn_ack = JSONArray[obj]['_source']['layers
    ']['tcp']['tcp.completeness_tree']['tcp.completeness.syn-ack']
173         except Exception:
174             tcp_completeness_syn_ack = None
175
176         try:
177             tcp_completeness_syn = JSONArray[obj]['_source']['layers']['
    tcp']['tcp.completeness_tree']['tcp.completeness.syn']
178         except Exception:
179             tcp_completeness_syn = None
180
181         try:
182             tcp_completeness_str = JSONArray[obj]['_source']['layers']['
    tcp']['tcp.completeness_tree']['tcp.completeness.str']
183             # if (tcp_completeness_str == "[ Null ]"): # TODO check if
    this won't break anything
184             #     tcp_completeness_str = None
185             tcp_completeness_str = ''.join(filter(str.isalnum,
```

```
    tcp_completeness_str))
186         except Exception:
187             tcp_completeness_str = None
188 
189         try:
190             tcp_flags_hex = JSONArray[obj]['_source']['layers']['tcp']['
    tcp.flags']
191             # if needed, adjust 010b and 10 to the desired length below
192             # see alternatives here: https://www.geeksforgeeks.org/
    python-ways-to-convert-hex-into-binary/
193             tcp_flags_bin = "{0:010b}".format(int(str(tcp_flags_hex),
    10))
194         except Exception:
195             tcp_flags_bin = None
196 
197         try:
198             tcp_flags_str = JSONArray[obj]['_source']['layers']['tcp']['
    tcp.flags_tree']['tcp.flags.str']
199             tcp_flags_str = ''.join(filter(str.isalnum, tcp_flags_str))
200         except Exception:
201             tcp_flags_str = None
202 
203         try:
204             tcp_window_size = JSONArray[obj]['_source']['layers']['tcp'
    ]['tcp.window_size']
205         except Exception:
206             tcp_window_size = None
207 
208         try:
209             tcp_window_size_scalefactor = JSONArray[obj]['_source']['
    layers']['tcp']['tcp.window_size_scalefactor']
210         except Exception:
211             tcp_window_size_scalefactor = None
212         # TCP only end #
213 
214         try:
215             frame_protocols = JSONArray[obj]['_source']['layers']['frame
    ']['frame.protocols']
216             frame_protocols = ''.join(filter(str.isalnum,
    frame_protocols))
217             # filter just erases all special chars, to keep the strings
    human readable, it's better to swap the chars instead
218             # frame_protocols = swap_special_chars_to_underscore(
    frame_protocols) # TODO test the performance of using this function
219         except Exception:
220             frame_protocols = None
221 
```

```
222         try:
223             ip_protocols = JSONArray[obj]['_source']['layers']['ip']['ip
    .proto']
224         except Exception:
225             ip_protocols = None
226 
227         try:
228             ip_flag_reserved_bit = JSONArray[obj]['_source']['layers']['
    ip']['ip.flags_tree']['ip.flags.rb']
229         except Exception:
230             ip_flag_reserved_bit = None
231 
232         try:
233             ip_flag_dont_fragment = JSONArray[obj]['_source']['layers']['
    ip']['ip.flags_tree']['ip.flags.df']
234         except Exception:
235             ip_flag_dont_fragment = None
236 
237         try:
238             ip_flag_more_fragments = JSONArray[obj]['_source']['layers'
    ]['ip']['ip.flags_tree']['ip.flags.mf']
239         except Exception:
240             ip_flag_more_fragments = None
241 
242         try:
243             ip_ttl = JSONArray[obj]['_source']['layers']['ip']['ip.ttl']
244         except Exception:
245             ip_ttl = None
246 
247         # TCP only begin #
248         try:
249             tcp_header_length = JSONArray[obj]['_source']['layers']['tcp
    ']['tcp.hdr_len']
250         except Exception:
251             tcp_header_length = None
252         # TCP only end #
253 
254         # UDP only begin #
255         try:
256             data_length = JSONArray[obj]['_source']['layers']['data']['
    data.len']
257         except Exception:
258             data_length = None
259         # UDP only end #
260 
261         # QUIC only begin #
262         try:
```

```
263             quic_packet_length = JSONArray[obj]['_source']['layers']['
    quic']['quic.packet_length']
264         except Exception:
265             quic_packet_length = None
266 
267         try:
268             quic_length = JSONArray[obj]['_source']['layers']['quic']['
    quic.length']
269         except Exception:
270             quic_length = None
271         # QUIC only end #
272 
273         record = [(pkt_number, timestamp, time_since_last_pkt,
    ipv4_ip_src, ipv4_ip_dst, frame_type, frame_len,
274                 header_len, payload_len, src_port, dst_port,
    tcp_completeness, tcp_completeness_reset,
275                 tcp_completeness_fin, tcp_completeness_data,
    tcp_completeness_ack, tcp_completeness_syn_ack,
276                 tcp_completeness_syn, tcp_completeness_str,
    tcp_flags_bin, tcp_flags_str, tcp_window_size,
277                 tcp_window_size_scalefactor, frame_protocols,
    ip_protocols, ip_flag_reserved_bit,
278                 ip_flag_dont_fragment, ip_flag_more_fragments, ip_ttl,
    tcp_header_length, data_length,
279                 quic_packet_length, quic_length)]
280 
281         df = pd.DataFrame.from_records(record, columns=labels)
282         with open(new_file_with_path, 'a', encoding='utf-8') as f:
283             # old way of reporting status, commented to avoiding
    flooding the stdout
284             # print("[INFO] Writing dataframe", obj, "of", length, "(",
    round(obj*100/length, 1) ,"% ) to CSV")
285             progressBar(obj, length, print_control, f"Writing {
    new_file_name}")
286             df.to_csv(f, header=False, index = False)
287 
288     print("\n[INFO] All", length, "records from", new_file_name, "were
    successfully written")
289     # print("[DEBU] Columns of CSV are", list(df)) # DEBUG
290 # parse("", "./") # Enable this line to run in "stand alone" mode (e.g.
    not importing as python module)
```

Source: Created by the author (2025).

APPENDIX K – Source of the box-plotter.py script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import pandas as pd
import matplotlib.pyplot as plt
import time
import os

from util import glob_get_files_list

# File paths
input_files_path = "./pcap/output/3-JSON-export/" # read CSV files from here
output_files_path = "./pcap/output/3-JSON-export/box-plots/" # save the output there

def plot_box_plot(file_path):
    # Read the CSV file
    df = pd.read_csv(file_path)
    input_file_name = os.path.splitext(file_path.split('/')[-1])[0]

    # If needed, drop some columns
    # df.drop(columns=["TCP_window_size"], axis=1, inplace=True) # drop TCP window size because of its size

    df.boxplot(vert=False, figsize=(6,8))
    plt.title(input_file_name + ' box plot')
    plt.xscale("symlog")
    # plt.subplots_adjust(left=0.28)
    plt.tight_layout()
    # Reference for the layout adjustment: https://stackoverflow.com/a/18500068

    full_output_path = output_files_path + input_file_name + '.pdf'
    plt.savefig(full_output_path, bbox_inches='tight', pad_inches=0, dpi=120)
    print("[INFO] Box plot sucessfully saved on", full_output_path)

    # plt.show() # DEBUG
    plt.close() # close each figure after finishing to enable multiple runs

start_time = time.time() # record the start of execution

# List of CSV files
```

```
csv_files = glob_get_files_list(input_files_path, "csv")

for i in csv_files:
    try:
        plot_box_plot(i)
    except AssertionError as e:
        print("[ERRO] Could not parse", i, "due to", e)
end_time = time.time() # record the end of execution
print(f"[DEBU] Execution time: {end_time - start_time} s")
```

Source: Created by the author (2025).

APPENDIX L – Source of the add_label_to_name.sh script

As indicated in Subsection **4.5.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
#!/usr/bin/env bash

# Helper script to iterate over all CSV files to add a label to their names
# this is necessary to help adding the labels to the data on the next step

CSV_FOLDER=./pcap/output/3-JSON-export/ # read CSV files from here
OUT_FOLDER=./pcap/output/4-ML/preprocess/labeled_files/ # save the output there
COUNTER=0

# List all CSV files in the input folder
FILES=("$CSV_FOLDER"*.csv)
FILES_LIST_SIZE=${#FILES[@]}

if [ $FILES_LIST_SIZE -eq 0 ]; then
    echo "[ERRO] No CSV files found in the directory: $CSV_FOLDER"
    exit 1
fi

echo "[INFO] Read $FILES_LIST_SIZE files"

# Loop through each file and rename it based on user input
for FILE in "${FILES[@]}"; do
    ((COUNTER++))

    # Extract the base name of the file (without extension)
    BASE_NAME=$(basename "$FILE" .csv)

    while true; do

        # Display options to the user
        echo "[INFO] Working on file $COUNTER of $FILES_LIST_SIZE"
        echo "Choose a label for $BASE_NAME :"
        echo "1. eMBB"
        echo "2. URLLC"
        echo "3. mMTC / mIoT"
        echo "4. Skip this file for now"
        echo "5. Cancel and exit"
        echo "" # new line to improve readability

```

```
40          read -p "Enter an option (1-5): " OPTION
41
42          # Validate the user's input
43          case "$OPTION" in
44              1)
45                  LABEL="embb"
46                  break
47                  ;;
48              2)
49                  LABEL="urllc"
50                  break
51                  ;;
52              3)
53                  LABEL="mmtc"
54                  break
55                  ;;
56              4)
57                  break
58                  ;;
59              5)
60                  echo "[INFO] User asked to quit"
61                  exit 1
62                  ;;
63              *)
64                  echo "[ERRO] Invalid option!"
65                  sleep 2
66                  ;;
67          esac
68      done
69
70      # Construct the new filename with the label appended
71      NEW_FILE="$OUT_FOLDER$BASE_NAME"_"$LABEL.csv"
72
73      # Move and rename the file
74      mv "$FILE" "$NEW_FILE"
75      # echo "[DEBU] Moved and renamed '$FILE' to '$NEW_FILE'" # DEBUG
76  done
77
78  echo "[INFO] All files have been processed"
```

Source: Created by the author (2025).

APPENDIX M – Source of the ml.py script

As indicated in Subsection **4.5.2**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import pandas as pd
import pickle
import csv
from numpy import mean,std
from datetime import datetime
from time import time_ns
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import HistGradientBoostingClassifier,
    RandomForestClassifier,AdaBoostClassifier,StackingClassifier,
    VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score,
                             precision_score,
                             recall_score,
                             f1_score,
                             roc_auc_score,
                             confusion_matrix,
                             make_scorer)

from util import (glob_get_files_list,
                  read_csv,
                  plot_confusion_matrix,
                  preprocess_data,
                  read_and_label_data,
                  data_oversample,
                  data_undersample,
                  generate_smaller_dataframe,
                  get_available_threads)

# File paths
working_folder = "./pcap/output/4-ML/"
input_files_path = working_folder + "preprocess/labeled_files/" # read
    CSV files from here
output_files_path_preprocessed_data = working_folder + "preprocess/
    data_ready_to_ml/" # save preprocessed data there
output_files_path_labeled_data = working_folder + "preprocess/
    labeled_data/" # save the labeled output files there
```

```
38 output_files_path_resampled_data = working_folder + "preprocess/
       resampled_data/" # save the labeled output files there
39 output_files_path_models = working_folder + "models/" # save the model
       files there
40 output_files_path_results = output_files_path_models + "training_results
       /" # save the model files there
41
42 # Control the execution of each function
43 run_model_training = True
44 run_cross_val = True #Cross validation
45 run_SMOTE = True # Oversampling
46 run_OSS = False # Undersampling
47
48 timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")
49 total_run_time_begin = datetime.now()
50
51 def classifier_select(classifier_acronym):
52     match classifier_acronym:
53         case 'LR':
54             clf = LogisticRegression()
55         case 'DT':
56             clf = DecisionTreeClassifier(max_depth=3)
57         case 'RF':
58             clf = RandomForestClassifier()
59         case 'MLP':
60             clf = MLPClassifier()
61         case 'SVM':
62             clf = LinearSVC()
63         case 'HGB':
64             clf = HistGradientBoostingClassifier()
65         case 'LightGBM':
66             clf = LGBMClassifier(verbose=-1)
67         case 'XGB':
68             clf = XGBClassifier()
69         case 'AdaBoost':
70             clf = AdaBoostClassifier()
71         case 'Stacking':
72             estimators = [
73             ('dt', DecisionTreeClassifier(max_depth=2)),
74             ('svc', LinearSVC()),
75             ('ada', AdaBoostClassifier())
76             ]
77             clf = StackingClassifier(estimators=estimators,
       final_estimator=LogisticRegression())
78         case 'Voting':
79             estimators = [
80             ('dt', DecisionTreeClassifier(max_depth=3)),
```

```
81              ('lr', LogisticRegression()),
82              ('ada', AdaBoostClassifier())
83              ]
84              clf = VotingClassifier(estimators=estimators, voting='soft',
    weights=[0.1, 0.1, 0.8])
85          case _:
86              print("[ERRO] Failed to load the model")
87              exit()
88
89      return clf
90
91  def save_model_locally(model, file_name, out_dir):
92      file_path = out_dir + file_name + ".pkl"
93      pickle.dump(model, open(file_path, 'wb'))
94      model_name = model.__class__.__name__
95      print(f"[INFO] Model {model_name} sucessfully saved on {file_path}")
96
97  # Load and prepare splits from labeled training data
98  def read_and_split_train_data(csv_file_list, split, dataset_percentage
    =100):
99      training_data = pd.DataFrame()
100     if (dataset_percentage == 100): # use the whole dataset
101         for file in csv_file_list:
102             if 'training' in file:
103                 df = read_csv(file)
104                 training_data = pd.concat([training_data, df],
    ignore_index=True)
105             elif 'inference' in file:
106                 # print("[DEBU] Skipped inference file found at", file)
    # DEBUG
107                 pass
108             elif 'SMOTE' in file and len(csv_file_list) == 1:
109                 training_data = read_csv(csv_file_list[0])
110             elif 'OSS' in file:
111                 training_data = read_csv(file)
112             else:
113                 raise ValueError(f"[ERRO] Could not determine data set
    type from filename {file}")
114                 exit()
115     else: # or a smaller portion of it (useful for testing the
    implementation)
116         training_data = generate_smaller_dataframe(csv_file_list,
    dataset_percentage)
117
118     # Prepare data for supervised learning
119     X = training_data.drop('label', axis=1)  # Features
120     y = training_data['label']  # Target variable
```

```
121
122     # print("\n[DEBU] Split data")
123     # print("[DEBU] Class distrib.:", y.value_counts()) # summarize
    class distribution
124     # print("[DEBU] Class distrib. (%):\n", y.value_counts(dropna=False,
    normalize=True)) # summarize class distribution
125     # print("[DEBU] Total no. training samples:", len(y))
126
127     if (split):
128         # Now X and y are ready for supervised learning
129         X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
130
131         return X_train, X_test, y_train, y_test
132     else:
133         return X, y
134
135 def train_models(model, X_train, X_test, y_train, y_test):
136     model_name = model.__class__.__name__
137     data_num_rows = len(X_train)
138
139     print("[INFO] Training model", model_name)
140     training_time_begin = time_ns()
141     model.fit(X_train, y_train)
142     training_time_end = time_ns()
143
144     training_disk_time_begin = time_ns()
145     save_model_locally(model, model_name, output_files_path_models)
146     training_disk_time_end = time_ns()
147
148     y_pred = model.predict(X_test)
149     cm = confusion_matrix(y_test, y_pred, labels=[0, 1, 2])
150     plot_confusion_matrix(cm, output_files_path_results, "", model_name,
    "training", timestamp, True, False)
151
152     # Model evaluation data
153     accuracy = accuracy_score(y_test, y_pred)
154     precision_avg = precision_score(y_test, y_pred, average="weighted")
155     recall_avg = recall_score(y_test, y_pred, average="weighted")
156     f_score_avg = f1_score(y_test, y_pred, average="weighted")
157     f_score_class0 = f1_score(y_test, y_pred, average=None, labels=[0])
    [0]
158     f_score_class1 = f1_score(y_test, y_pred, average=None, labels=[1])
    [0]
159     f_score_class2 = f1_score(y_test, y_pred, average=None, labels=[2])
    [0]
160
```

```
161     # print("[DEBU] Accuracy:", round(accuracy, 10))
162     # print(f"[DEBU] Confusion Matrix:\n{cm}")
163     # print("[DEBU] Precision :", round(precision_avg, 10))
164     # print("[DEBU] Recall    :", round(recall_avg, 10))
165     # print("[DEBU] F1-score  :", round(f_score_avg, 10))
166     # print("[DEBU] F1-score/class :", f1_score(y_test, y_pred, average=
    None, labels=[0, 1, 2]))
167     if (model_name != "LinearSVC" and model_name != "StackingClassifier"
    ): # LinearSVC doesn't implement proba
168         auc_score = roc_auc_score(y_test, model.predict_proba(X_test),
    average='macro', multi_class='ovo', labels=[0, 1, 2])
169         # print("[DEBU] ROC AUC Score :", round(auc_score, 10))
170     else:
171         # print("[DEBU] ROC AUC Score : Not calculated for", model_name)
172         auc_score = "N.A."
173 
174     if (model_name == 'DecisionTreeClassifier'):
175         # Record feature importance for Decision Tree
176         importance_with_columns = pd.DataFrame({'feature': X_train.
    columns, 'importance': model.feature_importances_})
177         importance_with_columns.sort_values(by='importance', ascending=
    False, inplace=True, ignore_index=True)
178         importance_with_columns.to_csv(f"{output_files_path_results}{
    timestamp}_dt_feature_importance.csv", header=True)
179 
180     elif (model_name == 'RandomForestClassifier'):
181         # Record feature importance for Random Forest
182         importance_with_columns = pd.DataFrame({'feature': X_train.
    columns, 'importance': model.feature_importances_})
183         importance_with_columns.sort_values(by='importance', ascending=
    False, inplace=True, ignore_index=True)
184         importance_with_columns.to_csv(f"{output_files_path_results}{
    timestamp}_rf_feature_importance.csv", header=True)
185 
186     training_time_ms = (training_time_end - training_time_begin) / 10**6
187     training_disk_time_ms = (training_disk_time_end -
    training_disk_time_begin) / 10**6
188     training_total_time_ms = training_time_ms + training_disk_time_ms
189 
190     new_row = {
191             columns_training_results_df[0]: model_name,
192             columns_training_results_df[1]: data_num_rows,
193             columns_training_results_df[2]: accuracy,
194             columns_training_results_df[3]: precision_avg,
195             columns_training_results_df[4]: recall_avg,
196             columns_training_results_df[5]: f_score_avg,
197             columns_training_results_df[6]: f_score_class0,
```

```
198             columns_training_results_df[7]: f_score_class1,
199             columns_training_results_df[8]: f_score_class2,
200             columns_training_results_df[9]: auc_score,
201             columns_training_results_df[10]: training_time_ms,
202             columns_training_results_df[11]: training_disk_time_ms,
203             columns_training_results_df[12]: training_total_time_ms,
204         }
205
206     # Add new row to the dataframe using loc[] method
207     training_results_df.loc[len(training_results_df)] = new_row
208
209     training_results_df.to_csv(f"{output_files_path_results}{timestamp}
    _training_results.csv", index=False)
210     print(f"[INFO] {model_name} training finished")
211
212 def cross_val(clf, X, y):
213     model_name = clf.__class__.__name__
214     data_num_rows = len(X)
215     print("[INFO] Running Cross Validation on", model_name)
216     scoring = {'prec_macro': 'precision_macro',
217             'rec_macro': make_scorer(recall_score, average='macro'),
218             'f1-score_avg': make_scorer(f1_score, average='weighted'),
219             # 'f1-score_class0': make_scorer(f1_score, average=None,
    labels=[0])[0]
220             }
221     cv_folds = 10 # reduce this number to reduce RAM usage during CV
222     # 3 uses up to around 50GB, 10 uses up to around 128GB (except for
    LinearSVC that requires more)
223     amount_of_cpus_to_use = 0.9 # e.g. if there are 16 CPUs available,
    90% will be equals to 14
224
225     # Run StratifiedKFold
226     scores = cross_validate(clf, X, y, scoring=scoring, cv=cv_folds,
    return_train_score=True, n_jobs=get_available_threads(
    amount_of_cpus_to_use))
227
228     s_fit_time_sec = scores['fit_time'] # time is in seconds for more
    information, see the URLs below
229     s_score_time_sec = scores['score_time'] # time is in seconds for
    more information, see the URLs below
230     # https://stackoverflow.com/questions/73548091/unit-for-fit-time-and
    -score-time-in-sklearn-cross-validate
231     # https://github.com/scikit-learn/scikit-learn/blob/55
    a65a2fa5653257225d7e184da3d0c00ff852b1/sklearn/model_selection/
    _validation.py#L673
232     s_train_precision = scores['train_prec_macro']
233     s_train_recall = scores['train_rec_macro']
```

```
234     s_train_f_score = scores['train_f1-score_avg']
235     s_test_precision = scores['test_prec_macro']
236     s_test_recall = scores['test_rec_macro']
237     s_test_f_score = scores['test_f1-score_avg']
238 
239     # print("[DEBU] Fit time:", s_fit_time)
240     # print("[DEBU] Score time:", s_score_time)
241     # print("[DEBU] -Train-")
242     # print("[DEBU] Precision:", s_train_precision)
243     # print("[DEBU] Recall:", s_train_recall)
244     # print("[DEBU] Average F1-Score:", s_train_f_score)
245     # print("[DEBU] -Test-")
246     # print("[DEBU] Precision:", s_test_precision)
247     # print("[DEBU] Recall:", s_test_recall)
248     # print("[DEBU] Average F1-Score:", s_test_f_score)
249 
250     avg_s_test_precision = mean(s_test_precision)
251     avg_s_test_recall = mean(s_test_recall)
252     avg_s_test_f_score = mean(s_test_f_score)
253     avg_s_fit_time_sec = mean(s_fit_time_sec)
254     avg_s_score_time_sec = mean(s_score_time_sec)
255 
256     cv_total_time_sec = s_fit_time_sec + s_score_time_sec
257 
258     new_row = {
259             columns_cross_val_results_df[0]: model_name,
260             columns_cross_val_results_df[1]: data_num_rows,
261             columns_cross_val_results_df[2]: mean(s_train_precision),
262             columns_cross_val_results_df[3]: mean(s_train_recall),
263             columns_cross_val_results_df[4]: mean(s_train_f_score),
264             columns_cross_val_results_df[5]: avg_s_test_precision,
265             columns_cross_val_results_df[6]: avg_s_test_recall,
266             columns_cross_val_results_df[7]: avg_s_test_f_score,
267             columns_cross_val_results_df[8]: std(s_test_precision, mean=
    avg_s_test_precision), # reuse the mean to improve performance
268             columns_cross_val_results_df[9]: std(s_test_recall, mean=
    avg_s_test_recall),       # see the URL below
269             columns_cross_val_results_df[10]: std(s_test_f_score, mean=
    avg_s_test_f_score),    # https://numpy.org/doc/stable/reference/
    generated/numpy.std.html
270             columns_cross_val_results_df[11]: avg_s_fit_time_sec,
271             columns_cross_val_results_df[12]: avg_s_score_time_sec,
272             columns_cross_val_results_df[13]: mean(cv_total_time_sec),
273             columns_cross_val_results_df[14]: std(s_fit_time_sec, mean=
    avg_s_fit_time_sec),  # reuse the mean to improve performance
274             columns_cross_val_results_df[15]: std(s_test_recall, mean=
    avg_s_score_time_sec), # see comment above
```

```
275         }
276 
277     # Add new row to the dataframe using loc[] method
278     cross_val_results_df.loc[len(cross_val_results_df)] = new_row
279 
280     cross_val_results_df.to_csv(f"{output_files_path_results}{timestamp}
    _cross_val_{cv_folds}_folds_results.csv", index=False)
281 
282     print(f"[INFO] {model_name} Cross Validation done")
283 
284 # Buid the CSV files list
285 csv_files = glob_get_files_list(input_files_path, file_format="csv")
286 
287 # Preprocess the data
288 preprocess_data(csv_files, output_files_path_preprocessed_data,
    output_files_path_results, timestamp, drop_protocols_and_ports=True)
289 
290 # Update the list of CSV files
291 csv_files = glob_get_files_list(output_files_path_preprocessed_data,
    file_format="csv")
292 
293 # Label all data inside CSV files
294 [read_and_label_data(file, output_files_path_labeled_data, False) for
    file in csv_files]
295 
296 # Update the list of CSV files
297 csv_files = glob_get_files_list(output_files_path_labeled_data,
    file_format="csv")
298 
299 # Prepare data splits to train the models
300 if (run_model_training or run_SMOTE or run_OSS):
301     print("[INFO] Preparing training data splits ... ", end='', flush=
    True)
302     if (run_OSS):
303         data_amount = 1 # amount of data to be used in OSS
304     else:
305         data_amount = 100
306     X_train, X_test, y_train, y_test = read_and_split_train_data(
    csv_files, split=True, dataset_percentage=data_amount)
307     print("[ OK ]")
308 
309 # Apply some data undersampling with OSS
310 if (run_OSS):
311     # OSS parameters
312     k = 1
313     seed = 100
314 
```

```
315     print("[INFO] Applying OSS on training data ... ", end='', flush=
    True)
316     OSS_run_time_begin = datetime.now()
317     X_train_oss, y_train_oss = data_undersample(X_train, y_train, k,
    seed)
318     print("[ OK ]")
319     # print("[DEBU] Data before OSS")
320     # print("[DEBU] Class distrib.:", y_train.value_counts()) #
    summarize class distribution
321     # print("[DEBU] Class distrib. (%):\n", y_train.value_counts(dropna=
    False, normalize=True)) # summarize class distribution
322     # print("[DEBU] Total no. training samples:", len(y_train))
323     # print("[DEBU] Data after OSS")
324     # print("[DEBU] Class distrib.:", y_train_oss.value_counts()) #
    summarize class distribution
325     # print("[DEBU] Class distrib. (%):\n", y_train_oss.value_counts(
    dropna=False, normalize=True)) # summarize class distribution
326     # print("[DEBU] Total no. training samples:", len(y_train_oss))
327
328     # Save the undersampled dataset on disk
329     undersampled_data = X_train_oss.join(y_train_oss)
330     undersampled_data.to_csv(f"{output_files_path_resampled_data}{
    timestamp}_OSS_undersample_k_{k}_seed_{seed}.csv", header=True, index
    =False)
331
332     # Use the undersampled data in the model training
333     X_train = X_train_oss
334     y_train = y_train_oss
335
336     OSS_run_time_end = datetime.now()
337     print(f"[INFO] Parameters: k = {k}, seed = {seed}")
338     print("[INFO] OSS run time:", (OSS_run_time_end - OSS_run_time_begin
    ).total_seconds(), "(seconds)")
339
340 # resampled_csv_files = glob_get_files_list(
    output_files_path_resampled_data, file_format="csv")
341 # TODO load the preprocessed files and use them to train the models
342
343 # Apply some data oversampling with SMOTE
344 if (run_SMOTE):
345     # SMOTE parameters
346     k = 5
347     strategy = 'minority'
348     # Update the list of CSV files
349     smote_files_available = glob_get_files_list(
    output_files_path_resampled_data, file_name_pattern=f"{k}*{strategy}"
    , file_format="csv", allow_empty_list=True)
```

```
350
351     if smote_files_available:
352         smote_files_available.sort()
353         smote_file = smote_files_available[-1] # get the most recent
    file
354
355         print(f"[INFO] File {smote_file} was found, skipping SMOTE")
356         print("[INFO] Reading preprocessed SMOTE file ... ", end='',
    flush=True)
357         X_train, X_test, y_train, y_test = read_and_split_train_data([
    str(smote_file)], split=True)
358         print("[ OK ]")
359
360     else:
361         print("[INFO] Applying SMOTE on training data ... ", end='',
    flush=True)
362         SMOTE_run_time_begin = datetime.now()
363         X_train_smote, y_train_smote = data_oversample(X_train, y_train,
     strategy=strategy, k=k)
364         print("[ OK ]")
365         # print("[DEBU] Data before SMOTE")
366         # print("[DEBU] Class distrib.:", y_train.value_counts()) #
    summarize class distribution
367         # print("[DEBU] Class distrib. (%):\n", y_train.value_counts(
    dropna=False, normalize=True)) # summarize class distribution
368         # print("[DEBU] Total no. training samples:", len(y_train))
369         # print("[DEBU] Data after SMOTE")
370         # print("[DEBU] Class distrib.:", y_train_smote.value_counts())
    # summarize class distribution
371         # print("[DEBU] Class distrib. (%):\n", y_train_smote.
    value_counts(dropna=False, normalize=True)) # summarize class
    distribution
372         # print("[DEBU] Total no. training samples:", len(y_train_smote)
    )
373         X_train = X_train_smote
374         y_train = y_train_smote
375
376         # Save the oversampled dataset on disk
377         oversampled_data = X_train_smote.join(y_train_smote)
378         oversampled_data.to_csv(f"{output_files_path_resampled_data}{
    timestamp}_SMOTE_oversample_k_{k}_strategy_{strategy}.csv", header=
    True, index=False)
379
380         SMOTE_run_time_end = datetime.now()
381         print("[INFO] SMOTE run time:", (SMOTE_run_time_end -
    SMOTE_run_time_begin).total_seconds(), "(seconds)")
382
```

```
383 model_names_list = ['LR', 'DT', 'RF', 'MLP', 'SVM', 'HGB', 'LightGBM', '
    XGB', 'AdaBoost', 'Stacking', 'Voting']
384
385 # Cross validation
386 if (run_cross_val):
387     columns_cross_val_results_df = ["model_name", "data_num_rows", "
    train_precision_avg", "train_recall_avg",
388                                     "train_f1_score_avg", "
    test_precision_avg", "test_recall_avg", "test_f1_score_avg",
389                                     "test_precision_stdev", "
    test_recall_stdev", "test_f1_score_stdev",
390                                     "fit_time_sec_avg", "score_time_sec_avg"
    , "total_cross_val_time_sec_avg",
391                                     "fit_time_sec_stdev", "
    score_time_sec_stdev"
392                                     ]
393     cross_val_results_df = pd.DataFrame(columns=
    columns_cross_val_results_df) # df to save the CV results
394
395     X, y = read_and_split_train_data(csv_files, split=False,
    dataset_percentage=100) # prepare data splits to cross val
396     for i in model_names_list:
397         clf = classifier_select(i)
398         cross_val(clf, X, y)
399
400 # Model training
401 if (run_model_training):
402     columns_training_results_df = ["model_name", "data_num_rows", "
    accuracy", "precision_avg", "recall_avg",
403                                    "f1_score_avg", "f1_score_class0", "
    f1_score_class1", "f1_score_class2",
404                                    "auc_score_avg", "training_time_ms", "
    training_disk_time_ms", "training_total_time_ms"]
405     training_results_df = pd.DataFrame(columns=
    columns_training_results_df) # df to save the training results
406     # Execute the actual model training
407     for i in model_names_list:
408         clf = classifier_select(i)
409
410         train_models(clf, X_train, X_test, y_train, y_test)
411
412 total_run_time_end = datetime.now()
413 print("[INFO] ML total run time:", (total_run_time_end -
    total_run_time_begin).total_seconds(), "(seconds)")
```

Source: Created by the author (2025).

APPENDIX N – Source of the inference.py script

As indicated in Subsection **4.5.2**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import pandas as pd
import numpy as np
import pickle
from datetime import datetime
from time import time_ns
from sklearn.metrics import (accuracy_score,
                             precision_score,
                             recall_score,
                             f1_score,
                             roc_auc_score,
                             confusion_matrix)

from util import glob_get_files_list,read_csv,label_id_to_text,
    plot_confusion_matrix,read_and_label_data,preprocess_data

# File paths
working_folder = "./pcap/output/4-ML/"
input_files_path = working_folder + "preprocess/labeled_files/" # read
    labeled data CSV files from here
output_files_path_preprocessed_data = working_folder + "preprocess/
    data_ready_to_ml/" # save preprocessed data there
models_folder = working_folder + "models/" # read the models from here
data_folder = working_folder + "preprocess/labeled_data/" # save or read
     the inference data from here
results_folder = working_folder + "inference_results/" # save the
    results here

timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")

def run_inference(models_file_list, inference_data_file_list, extra_run=
    False):
    for file in inference_data_file_list:
        file_name = file.split('/')[-1]

        if 'inference' in file:
            print("[INFO] Running inference on", file_name)

            columns = ["file_name", "file_num_rows", "model_name",
                       "inference_result_label", "inference_result",
                       "accuracy", "precision", "recall", "
    f_score_class",
                       "inference_result_count_0", "
```

```
    inference_result_count_1", "inference_result_count_2",
36                      "inf_disk_time_ms", "inf_pred_time_ms", "
    inf_total_time_ms"]
37          # Initialize results_df with columns
38          results_df = pd.DataFrame(columns=columns)
39
40          data = read_csv(file) # load inference data
41          data_num_rows = len(data)
42
43          for path in models_file_list:
44              total_inference_time_begin = time_ns()
45              model_load_time_begin = time_ns()
46              model = pickle.load(open(path, 'rb')) # load model from
    disk
47              model_load_time_end = time_ns() # TODO measure and save
    it
48              model_name = model.__class__.__name__
49              print("[INFO] Using", model_name)
50
51              y_true = data['label'] # save the labels for evaluation
52              true_label = y_true.iloc[0] # save true label sample for
     evaluation
53              inference_data = data.drop('label', axis=1)  # remove
    the label column
54
55              inference_pred_time_begin = time_ns()
56              y_pred = model.predict(inference_data)
57              inference_pred_time_end = time_ns()
58              inference_result = pd.Series([model.classes_[i] for i in
     y_pred])
59              inference_result_counts = inference_result.value_counts
    ()
60              inference_result_int = inference_result_counts.idxmax()
61              inference_result_label = label_id_to_text(
    inference_result_int)
62              total_inference_time_end = time_ns()
63              # print(f"[DEBU] Inference result: {inference_result_int
    } ({inference_result_label})")  # DEBUG
64              # print(f"[DEBU] Labels and their occurrences:\n{
    inference_result_counts}")  # DEBUG
65
66              total_inference_time = (total_inference_time_end -
    total_inference_time_begin) / 10**6
67              inference_pred_time = (inference_pred_time_end -
    inference_pred_time_begin) / 10**6
68              inference_disk_time = (model_load_time_end -
    model_load_time_begin) / 10**6
```

```
69
70                  # Model evaluation data
71                  accuracy = accuracy_score(y_true, y_pred)
72                  cm = confusion_matrix(y_true, y_pred, labels=[0, 1, 2])
73                  precision = precision_score(y_true, y_pred, average="
    weighted", zero_division=np.nan, labels=[true_label])
74                  recall = recall_score(y_true, y_pred, average="weighted"
    , zero_division=np.nan)
75                  f_score_class = f1_score(y_true, y_pred, average="
    weighted", zero_division=np.nan, labels=[true_label])
76                  plot_confusion_matrix(cm, results_folder, file_name,
    model_name, "inference", timestamp, True, False)
77
78                  # print("[DEBU] Accuracy:", round(accuracy, 10))
79                  # print(f"[DEBU] Confusion Matrix:\n{cm}")
80                  # print("[DEBU] Precision :", round(precision, 10))
81                  # print("[DEBU] Recall    :", round(recall, 10))
82                  # print("[DEBU] F1-score of class :", round(
    f_score_class, 10))
83                  # if (model_name != "LinearSVC"): # LinearSVC doesn't
    implement proba
84                  #     auc_score = roc_auc_score(y_true, model.
    predict_proba(inference_data), average='macro', multi_class='ovo',
    labels=[0, 1, 2])
85                  #     print("[DEBU] ROC AUC Score :", round(auc_score,
    10))
86                  # else:
87                  #     print("[DEBU] ROC AUC Score : Not calculated for",
     model_name)
88                  #     auc_score = "N.A."
89                  # TODO fix ROC AUC score
90                  # RuntimeWarning: invalid value encountered in scalar
    divide ret = ret.dtype.type(ret / rcount) /n [DEBU] ROC AUC Score :
    nan
91
92                  new_row = {
93                      columns[0]: file_name,
94                      columns[1]: data_num_rows,
95                      columns[2]: model_name,
96                      columns[3]: inference_result_label,
97                      columns[4]: inference_result_int,
98                      columns[5]: accuracy,
99                      columns[6]: precision,
100                     columns[7]: recall,
101                     columns[8]: f_score_class,
102                     columns[9]: int(inference_result_counts[0]) if 0 in
    inference_result_counts else 0,
```

```
103                     columns[10]: int(inference_result_counts[1]) if 1 in
    inference_result_counts else 0,
104                     columns[11]: int(inference_result_counts[2]) if 2 in
    inference_result_counts else 0,
105                     columns[12]: inference_disk_time,
106                     columns[13]: inference_pred_time,
107                     columns[14]: total_inference_time,
108                 }
109 
110                 # Add new row to the dataframe using loc[] method
111                 results_df.loc[len(results_df)] = new_row
112 
113             # Get the original input file name without format
114             output_filename = file_name.split('/')[-1].split('
    _inference_')[0]
115             # Save the results dataframe to a CSV file with a unique
    filename based on the current time
116             if not extra_run:
117                 results_df.to_csv(f"{results_folder}{output_filename}_{
    timestamp}_inference_results.csv", index=False)
118             else:
119                 # adjust parameters if running multiple times
120                 results_df.to_csv(f"{results_folder}{output_filename}_{
    timestamp}_inference_results.csv", index=False, mode='a', header=
    False)
121 
122         else:
123             print(f"[WARN] Skipping file {file_name}")
124 
125 print("[INFO] Running inference preprocess")
126 # Run preprocess in case the inference data wasn't already preprocessed
127 total_preprocess_time_begin = datetime.now()
128 # Get the list of CSV files
129 csv_files = glob_get_files_list(input_files_path, file_format="csv")
130 
131 # Preprocess the data
132 preprocess_data(csv_files, output_files_path_preprocessed_data,
    results_folder, timestamp, drop_protocols_and_ports=True)
133 
134 # Update the list of CSV files
135 csv_files = glob_get_files_list(output_files_path_preprocessed_data,
    file_format="csv")
136 
137 # Label all data inside CSV files
138 [read_and_label_data(file, data_folder, False) for file in csv_files]
139 total_preprocess_time_end = datetime.now()
140 
```

```
141 print("[INFO] Running inference")
142 # Buid the PKL and CSV files lists
143 pkl_files = glob_get_files_list(models_folder, file_format="pkl")
144 inference_data_files = glob_get_files_list(data_folder, file_format="csv
    ")
145
146 total_run_time_begin = datetime.now()
147 run_inference(pkl_files, inference_data_files)
148 run_inference(pkl_files, inference_data_files, extra_run=True)
149 run_inference(pkl_files, inference_data_files, extra_run=True)
150 total_run_time_end = datetime.now()
151 print("[INFO] Total preprocess run time:", (total_preprocess_time_end -
    total_preprocess_time_begin).total_seconds(), "(seconds)")
152 print("[INFO] Total inference run time:", (total_run_time_end -
    total_run_time_begin).total_seconds(), "(seconds)")
```

Source: Created by the author (2025).

APPENDIX O – Source of the traffic generator scripts

As indicated in Subsection **4.5.3**, these scripts are part of the implemented `NWDAF_ml` module. The ready-to-use source code and documentation are available in a public GitHub repository (41).

The `play-video.sh` script allows for playing stored videos or streaming live broadcasts depending on the selected mode. As detailed below, the URLs can be easily customized to use other online services, and may use Mozilla Firefox or Brave internet browsers to perform the configured task.

```
#!/usr/bin/env bash

IP_UE=10.60.0.1 # IP given by 5GC to point-to-point interface (a.k.a. uesimtun0)
#YT_URL="https://www.youtube.com/watch?v=LXb3EKWsInQ" # URL of the stored video
YT_URL="https://www.youtube.com/watch?v=LXb3EKWsInQ&list=PLrN5hDSKBQCLN_p4SJwqHSNO3ToomP-il&index=1" # URL of a ~2h playlist of stored videos
NAVER_TV_URL="https://tv.naver.com/l/164367" # URL of the live stream
PREPARE_MODE=0
RUN_MODE=0
LIVE_STREAM=0 # If 1, will use YT_URL, if 0, will use NAVER_TV_URL
USE_FIREFOX=1 # If 1, will use Mozilla Firefox, if 0, will use Brave Browser

# check the input parameters and set the control vars accordingly
if [ $# -gt 0 ]; then
    while [ $# -gt 0 ]; do
        case $1 in
        -prepare)
            PREPARE_MODE=1
            ;;
        -play-yt)
            RUN_MODE=1
            LIVE_STREAM=0
            ;;
        -play-live)
            RUN_MODE=1
            LIVE_STREAM=1
            ;;
        -use-brave)
            if [ $# -gt 1 ]; then
                USE_FIREFOX=0
            else
                echo "[ERRO] The option '-use-brave' can't be used alone
```

```
    "
                exit 1
            fi
            ;;
        *)
            echo "[ERRO] Some input parameter wasn't found. Check your
    input and try again"
            exit 1
            ;;
        esac
        shift
    done
else
    echo "[ERRO] At least one parameter is required"
    exit 1
fi

check_x_server_availability () {
    # forwarding is necessary to run the browser correctly
    # when connecting remotely
    if [ -n "$DISPLAY" ]; then
        echo "[INFO] X11 forwarding is enabled."
    else
        echo "[ERRO] X11 forwarding is not enabled. Connect to SSH using
    -X parameter"
        exit 1
    fi
}

prepare () {
    if [[ $USE_FIREFOX -eq 1 ]]; then
        sudo apt install firefox
    elif [[ $USE_FIREFOX -eq 0 ]]; then
        # Install Brave browser instead
        curl -fsS https://dl.brave.com/install.sh | sh
    fi
}

run () {
    if [ ! -f ./nr-binder ]; then
        echo "[ERRO] nr-binder not found!"
        echo "TIP: move $0 to the same folder as nr-binder"
        # currently this folder is called 'build' on UERANSIM sources
    fi

    # Set URL to be played
    if [[ $LIVE_STREAM -eq 0 ]]; then
```

```
76          URL=$YT_URL
77      elif [[ $LIVE_STREAM -eq 1 ]]; then
78          URL=$NAVER_TV_URL
79      fi
80
81      if [[ $USE_FIREFOX -eq 1 ]]; then
82          bash nr-binder $IP_UE firefox --new-window $URL
83      elif [[ $USE_FIREFOX -eq 0 ]]; then
84          # Use Brave browser as an alternative
85          bash nr-binder $IP_UE brave-browser --new-window --incognito
    $URL
86          # using incognito to prevent Brave resuming a previous session
87          # (i.e. 'Continue where you left off')
88      fi
89  }
90
91  check_x_server_availability
92
93  if [[ $PREPARE_MODE -eq 0 && $RUN_MODE -eq 1 ]]; then
94      run
95  elif [[ $PREPARE_MODE -eq 1 && $RUN_MODE -eq 0 ]]; then
96      prepare
97  elif [[ $PREPARE_MODE -eq 1 && $RUN_MODE -eq 1 ]]; then
98      prepare
99      run
100 fi
```

Source: Created by the author (2025).

The `udp-server.sh` and `udp-client.sh` scripts facilitate the transmission of UDP packets at a fixed rate or within a probability-based interval range. As detailed below, these scripts provide customization while maintaining simplicity.

```
1 #!/usr/bin/env bash
2
3 # Basic UDP server
4
5 PORT=30000 # port used to listen for client packets
6
7 echo "[INFO] Starting server"
8 nc -u -k -l $PORT # nc server
```

Source: Created by the author (2025).

```
#!/usr/bin/env bash

# Basic UDP client

# SLEEP_TIMER=60 # send packets in a fixed rate (1 packet/sec)
# SLEEP_TIMER=0.01 # send packets in a fixed rate (100 packets/sec)
IP_UE=10.60.0.1
IP_5GC=10.0.0.110
DEST_PORT=30000 # port used to connect to server
SOURCE_PORT=1337 # port used to send the packets

get_sleep_time(){
    # Generate a random number from 0 to 100
    num=$(shuf -i 0-100 -n 1)

    # Determine which range the number falls into based on the probabilities
    # the intervals below were based on https://doi.org/10.1109/INFCOMW.2017.8116438 (page 3)
    if [ $num -lt 85 ]; then # 0-20 range (85% probability)
            min=1
            max=20
    elif [ $num -lt 96 ]; then # 20-60 range (11% probability)
            min=20
            max=60
    else # 60-90 range (the 4% probability left)
            min=60
            max=90
            # NOTE source above doesn't specify a upper limit
            # it only says "longer than one minute"
            # so I decided to limit to 90s
    fi

    raffle=$((min + RANDOM % (max - min + 1)))

    # echo "[DEBU] Value: $raffle"
    echo "$raffle"
}

while true; do
    # SLEEP_TIMER=$((1 + RANDOM % 20)) # send packets in the interval between 1 to 20 seconds
    SLEEP_TIMER=$(get_sleep_time) # send packets in the interval between 1 to 90 seconds with probabilities
    LOAD_ONE_MIN=$(cat /proc/loadavg | awk '{print $1}')
    echo "[INFO] Sending data"
    echo -e "System Load: $LOAD_ONE_MIN Next update in $SLEEP_TIMER
```

```
    seconds" | nc -4 -u -w0 -s $IP_UE $IP_5GC $DEST_PORT -p $SOURCE_PORT
    # nc client
44   echo "[INFO] Sleeping for $SLEEP_TIMER seconds..."
45   sleep $SLEEP_TIMER
46 done
```

Source: Created by the author (2025).

APPENDIX P – Source of the model_tuning.py script

As indicated in Subsection **5.2.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import numpy as np
import pandas as pd
from sklearn.metrics import make_scorer, f1_score
from sklearn.model_selection import RepeatedStratifiedKFold,
    cross_val_score,StratifiedKFold
from scipy.optimize import differential_evolution
from util import read_csv,glob_get_files_list

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import HistGradientBoostingClassifier,
    RandomForestClassifier,AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier

# File paths
working_folder = "./pcap/output/4-ML/"
input_files_path = working_folder + "preprocess/labeled_data/" # read
    CSV files from here

def hyperparam_tuning_DE(params, params_names, estimator, X_train,
    y_train):
     model_name = estimator.__class__.__name__
     params_dict = {params_names[i]:params[i] for i in range(len(params))
    }

     try:
         match model_name:
             case "LogisticRegression":
                 penalty = ['l1', 'l2', 'elasticnet', None]
                 solver = ['lbfgs', 'newton-cg', 'newton-cholesky', 'sag'
    , 'saga']

                 params_dict["penalty"] = penalty[int(params_dict["
    penalty"])]
                 params_dict["solver"] = solver[int(params_dict["solver"
    ])]

                 params_dict["max_iter"] = int(params_dict["max_iter"]) #
     cast param to int type
```

```
34
35          case "LinearSVC":
36              penalty = ['l1', 'l2']
37              loss = ['hinge', 'squared_hinge']
38
39              params_dict["penalty"] = penalty[int(params_dict["
    penalty"])]
40              params_dict["loss"] = loss[int(params_dict["loss"])]
41
42              params_dict["max_iter"] = int(params_dict["max_iter"]) #
     cast param to int type
43
44          case "HistGradientBoostingClassifier":
45              params_dict["max_iter"] = int(params_dict["max_iter"]) #
     cast param to int type
46              params_dict["max_leaf_nodes"] = int(params_dict["
    max_leaf_nodes"]) # cast param to int type
47              params_dict["max_depth"] = int(params_dict["max_depth"])
     # cast param to int type
48              params_dict["min_samples_leaf"] = int(params_dict["
    min_samples_leaf"]) # cast param to int type
49              params_dict["max_bins"] = int(params_dict["max_bins"]) #
     cast param to int type
50
51          case "RandomForestClassifier":
52              criterion = ["gini", "entropy", "log_loss"]
53              max_features = ["sqrt", "log2", None]
54
55              params_dict["criterion"] = criterion[int(params_dict["
    criterion"])]
56              params_dict["max_features"] = max_features[int(
    params_dict["max_features"])]
57
58              params_dict["n_estimators"] = int(params_dict["
    n_estimators"]) # cast param to int type
59              params_dict["max_depth"] = int(params_dict["max_depth"])
     # cast param to int type
60              params_dict["min_samples_split"] = int(params_dict["
    min_samples_split"]) # cast param to int type
61              params_dict["min_samples_leaf"] = int(params_dict["
    min_samples_leaf"]) # cast param to int type
62              params_dict["max_leaf_nodes"] = int(params_dict["
    max_leaf_nodes"]) # cast param to int type
63
64          case "DecisionTreeClassifier":
65              criterion = ["gini", "entropy", "log_loss"]
66              splitter = ["best", "random"]
```

```
67                  max_features = ["sqrt", "log2", None]
68
69                  params_dict["criterion"] = criterion[int(params_dict["
    criterion"])]
70                  params_dict["splitter"] = splitter[int(params_dict["
    splitter"])]
71                  params_dict["max_features"] = max_features[int(
    params_dict["max_features"])]
72
73                  params_dict["max_depth"] = int(params_dict["max_depth"])
     # cast param to int type
74                  params_dict["min_samples_split"] = int(params_dict["
    min_samples_split"]) # cast param to int type
75                  params_dict["min_samples_leaf"] = int(params_dict["
    min_samples_leaf"]) # cast param to int type
76                  params_dict["max_leaf_nodes"] = int(params_dict["
    max_leaf_nodes"]) # cast param to int type
77
78              case "MLPClassifier":
79                  hidden_layer_sizes = ["(100,)", "(100, 50, 25)", "(100,
    100, 50)"]
80                  activation = ["identity", "logistic", "tanh", "relu"]
81                  solver = ["lbfgs", "sgd", "adam"]
82
83                  params_dict["hidden_layer_sizes"] = hidden_layer_sizes[
    int(params_dict["hidden_layer_sizes"])]
84                  params_dict["activation"] = activation[int(params_dict["
    activation"])]
85                  params_dict["solver"] = solver[int(params_dict["solver"
    ])]
86
87                  params_dict["max_iter"] = int(params_dict["max_iter"]) #
     cast param to int type
88
89              case "LGBMClassifier":
90                  params_dict["num_leaves"] = int(params_dict["num_leaves"
    ]) # cast param to int type
91                  params_dict["max_depth"] = int(params_dict["max_depth"])
     # cast param to int type
92                  params_dict["n_estimators"] = int(params_dict["
    n_estimators"]) # cast param to int type
93                  params_dict["subsample_for_bin"] = int(params_dict["
    subsample_for_bin"]) # cast param to int type
94                  params_dict["min_child_samples"] = int(params_dict["
    min_child_samples"]) # cast param to int type
95
96              case "XGBClassifier":
```

```
97                  params_dict["eta"] = int(params_dict["eta"]) # cast
    param to int type
98                  params_dict["max_depth"] = int(params_dict["max_depth"])
     # cast param to int type
99                  params_dict["min_child_weight"] = int(params_dict["
    min_child_weight"]) # cast param to int type
100                 params_dict["max_delta_step"] = int(params_dict["
    max_delta_step"]) # cast param to int type
101                 params_dict["max_leaves"] = int(params_dict["max_leaves"
    ]) # cast param to int type
102                 params_dict["max_bin"] = int(params_dict["max_bin"]) #
    cast param to int type
103
104             case "AdaBoostClassifier":
105                 params_dict["n_estimators"] = int(params_dict["
    n_estimators"]) # cast param to int type
106
107             case _:
108                 print("[ERRO] Failed to load the model")
109                 exit()
110
111         print("Params:", params_dict)
112
113         estimator.set_params(**params_dict)
114
115         # cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
    random_state=1)
116         cv = StratifiedKFold(n_splits=3)
117
118         cv_score = -1 * np.mean(cross_val_score(estimator, X_train,
    y_train, cv=cv, scoring=make_scorer(f1_score, average='weighted')))
119
120     except Exception as e:
121         # print(e)
122         cv_score = 0
123
124     print(cv_score)
125
126     return cv_score
127
128 def classifier_select(classifier_acronym):
129     match classifier_acronym:
130         case 'LR':
131             clf = LogisticRegression()
132         case 'DT':
133             clf = DecisionTreeClassifier()
134         case 'RF':
```

```
135             clf = RandomForestClassifier()
136         case 'MLP':
137             clf = MLPClassifier()
138         case 'SVM':
139             clf = LinearSVC()
140         case 'HGB':
141             clf = HistGradientBoostingClassifier()
142         case 'LightGBM':
143             clf = LGBMClassifier(verbose=-1)
144         case 'XGB':
145             clf = XGBClassifier()
146         case 'AdaBoost':
147             clf = AdaBoostClassifier()
148 
149     return clf
150 
151 def run_model_tuning(clf, X, y):
152     model_name = clf.__class__.__name__
153     print("Running for", model_name)
154 
155     # Mapping of model names to their parameters and integrality
156     models = {
157         "LogisticRegression": (lr_params, lr_integrality),
158         "LinearSVC": (svc_params, svc_integrality),
159         "HistGradientBoostingClassifier": (hgb_params, hgb_integrality),
160         "RandomForestClassifier": (rf_params, rf_integrality),
161         "DecisionTreeClassifier": (dt_params, dt_integrality),
162         "MLPClassifier": (mlp_params, mlp_integrality),
163         "LGBMClassifier": (lgbm_params, lgbm_integrality),
164         "XGBClassifier": (xgb_params, xgb_integrality),
165         "AdaBoostClassifier": (adaboost_params, adaboost_integrality)
166     }
167 
168     # Get the parameters and integrality for the specified model
169     if model_name in models:
170         params, integrality = models[model_name]
171         result = differential_evolution(hyperparam_tuning_DE, list(list(
    params.values())), args=(list(params.keys()), clf, X, y), maxiter=4,
    popsize=4, tol=0.01, workers=10, integrality=integrality, disp=True,
    updating='deferred')
172         print("Dict:", result)
173         print("Best params:", result.x)
174     else:
175         print(f"[ERRO] Model {model_name} not found.")
176 
177 # Load and prepare splits from labeled training data
178 def read_and_split_train_data(csv_file_list, split, dataset_percentage
```

```
    =100):
179     training_data = pd.DataFrame()
180     if (dataset_percentage == 100): # use the whole dataset
181         for file in csv_file_list:
182             if 'training' in file:
183                 df = read_csv(file)
184                 training_data = pd.concat([training_data, df],
    ignore_index=True)
185             elif 'inference' in file:
186                 # print("[DEBU] Skipped inference file found at", file)
    # DEBUG
187                 pass
188             else:
189                 raise ValueError(f"[ERRO] Could not determine data set
    type from filename {file}")
190                 exit()
191     else: # or a smaller portion of it (useful for testing the
    implementation)
192         training_data = generate_smaller_dataframe(csv_file_list,
    dataset_percentage)
193
194     # Prepare data for supervised learning
195     X = training_data.drop('label', axis=1)  # Features
196     y = training_data['label']  # Target variable
197
198     print("[DEBU] Split data")
199     print("[DEBU] Class distrib.:", y.value_counts()) # summarize class
    distribution
200     print("[DEBU] Class distrib. (%):\n", y.value_counts(dropna=False,
    normalize=True)) # summarize class distribution
201     print("[DEBU] Total no. training samples:", len(y))
202     # exit()
203
204     if (split):
205         # Now X and y are ready for supervised learning
206         X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
207
208         return X_train, X_test, y_train, y_test
209     else:
210         return X, y
211
212 # Parameters to be tested
213 # Logistic Regression
214 lr_integrality = [0, 1, 1, 1]
215 lr_params = {
216             'C': [0.01, 100.0],
```

```
217             'penalty': [0, 3],
218             'solver': [0, 4],
219             'max_iter': [100, 1000]
220            }
221 # For more LR parameters check: https://scikit-learn.org/stable/modules/
    generated/sklearn.linear_model.LogisticRegression.html
222
223 # Linear Support Vector Machine
224 svc_integrality = [0, 1, 1, 0, 1]
225 svc_params = {
226              'C': [0.01, 100.0],
227              'penalty': [0, 1],
228              'loss': [0, 1],
229              'intercept_scaling': [1.0, 2.0],
230              'max_iter': [1000, 5000]
231             }
232 # For more LinearSVC parameters check: https://scikit-learn.org/stable/
    modules/generated/sklearn.svm.LinearSVC.html
233
234 # Histogram-based Gradient Boosting Classification Tree
235 hgb_integrality = [0, 1, 1, 1, 1, 0, 0, 1]
236 hgb_params = {
237              'learning_rate': [0.01, 1.0],
238              'max_iter': [100, 1000],
239              'max_leaf_nodes': [1, 100],
240              'max_depth': [3, 50],
241              'min_samples_leaf': [10, 100],
242              'l2_regularization': [0.0, 1.0],
243              'max_features': [1.0, 10.0],
244              'max_bins': [100, 1000]
245             }
246 # For more LinearSVC parameters check: https://scikit-learn.org/stable/
    modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.
    html
247
248 # Random Forest
249 rf_integrality = [1, 1, 1, 1, 1, 1, 1, 0]
250 rf_params = {
251             'n_estimators': [50, 1000],
252             'criterion': [0, 2],
253             'max_depth': [3, 20],
254             'min_samples_split': [1000, 100000],
255             'min_samples_leaf': [1000, 100000],
256             'max_features': [0, 2],
257             'max_leaf_nodes': [1, 50],
258             'min_impurity_decrease': [0.0, 0.8],
259            }
```

```
# For more RandomForest parameters check: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

# Decision Tree
dt_integrality = [1, 1, 1, 1, 1, 1, 0, 0]
dt_params = {
             'criterion': [0, 2],
             'splitter': [0, 1],
             'max_depth': [3, 20],
             'min_samples_split': [1000, 100000],
             'min_samples_leaf': [1000, 100000],
             'max_features': [0, 2],
             'max_leaf_nodes': [1, 50],
             'min_impurity_decrease': [0.0, 0.8],
             'ccp_alpha': [0.0, 1.0]
            }
# For more DecisionTreeClassifier parameters check: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

# Multilayer Perceptron
mlp_integrality = [1, 1, 1, 0, 0, 1, 0]
mlp_params = {
              'hidden_layer_sizes': [0, 2],
              'activation': [0, 3],
              'solver': [0, 2],
              'alpha': [0.0001, 10.0],
              'learning_rate_init': [0.001, 0.1],
              'max_iter': [200, 1000],
              'tol': [0.0001, 0.01]
              }
# For more MLPClassifier parameters check: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

# Light GBM
lgbm_integrality = [1, 1, 0, 1, 1, 0, 0, 1, 0, 0]
lgbm_params = {
               'num_leaves': [20, 100],
               'max_depth': [-1, 10],
               'learning_rate': [0.01, 1.0],
               'n_estimators': [50, 200],
               'subsample_for_bin': [100000, 300000],
               'min_split_gain': [0.01, 0.8],
               'min_child_weight': [0.001, 0.1],
               'min_child_samples': [1000, 10000],
               'subsample': [0.5, 1.0],
               'colsample_bytree': [0.5, 1.0]
```

```
303                 }
304 # For more LGBMClassifier parameters check: https://lightgbm.readthedocs
    .io/en/stable/pythonapi/lightgbm.LGBMClassifier.html
305
306 # eXtreme Gradient Boosting
307 xgb_integrality = [0, 0, 1, 1, 1, 0, 0, 0, 1, 1]
308 xgb_params = {
309                 'eta': [0.0, 1.0],
310                 'gamma': [0.0, 1.0],
311                 'max_depth': [5, 20],
312                 'min_child_weight': [1, 10000],
313                 'max_delta_step': [0, 10],
314                 'subsample': [0.5, 1.0],
315                 'lambda': [0.0, 2.0],
316                 'alpha': [0.0, 2.0],
317                 'max_leaves': [0, 100],
318                 'max_bin': [128, 512]
319               }
320 # For more XGBClassifier parameters check: https://xgboost.readthedocs.
    io/en/stable/parameter.html
321
322 # AdaBoost
323 adaboost_integrality = [1, 0]
324 adaboost_params = {
325                     'n_estimators': [10, 1000],
326                     'learning_rate': [0.0, 5.0],
327                   }
328 # For more AdaBoost parameters check: https://scikit-learn.org/stable/
    modules/generated/sklearn.ensemble.AdaBoostClassifier.html
329
330 model_names_list = ['LR', 'DT', 'RF', 'MLP', 'SVM', 'HGB', 'LightGBM', '
    XGB', 'AdaBoost']
331
332 # Buid the CSV files list
333 csv_files = glob_get_files_list(input_files_path, file_format="csv")
334
335 X, y = read_and_split_train_data(csv_files, split=False,
    dataset_percentage=100) # prepare data splits to tuning
336
337 # Run the tuning for all models in the list
338 for i in model_names_list:
339         clf = classifier_select(i)
340         run_model_tuning(clf, X, y)
```

Source: Created by the author (2025).

APPENDIX Q – Source of the dt_visualization.py script

As indicated in Subsection **5.2.1**, this script is part of the `NWDAF_ml` module. The ready-to-use source code and documentation are available in a GitHub repository (41).

```
import pandas as pd
import time
from subprocess import check_call
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.model_selection import train_test_split
from util import (glob_get_files_list,
                  read_csv,
                  data_oversample)

# File paths
working_folder = "./pcap/output/4-ML/"
input_files_path_labeled_data = working_folder + "preprocess/
    labeled_data/" # read the labeled output files from here
# Vars
class_names_list = ["eMBB","URLLC","mMTC"]
features_names_list = []
start_time = time.time() # record the start of execution

# Load and prepare splits from labeled training data
def read_and_split_train_data(csv_file_list, split, with_features_names=
    False):
    training_data = pd.DataFrame()
    for file in csv_file_list:
        if 'training' in file:
            df = read_csv(file)
            training_data = pd.concat([training_data, df], ignore_index=
    True)
        elif 'inference' in file:
            # print("[DEBU] Skipped inference file found at", file) #
    DEBUG
            pass
        else:
            raise ValueError(f"[ERRO] Could not determine data set type
    from filename {file}")
            exit()

    # Prepare data for supervised learning
    X = training_data.drop('label', axis=1)  # Features
    y = training_data['label']  # Target variable

    if (split):
        # Now X and y are ready for supervised learning
```

```
38          X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
39
40          if (with_features_names):
41              feature_list = list(X.columns)
42
43              return X_train, X_test, y_train, y_test, feature_list
44          else:
45              return X_train, X_test, y_train, y_test
46
47      else:
48          return X, y
49
50 # Create the DT visualization plots
51 def create_and_plot_tree_visualization(file_name_suffix,
    rounded_rectangles, horizontal, print_percentages, parallel_leaves=
    False):
52      file_path_without_format = "img/dtree_" + file_name_suffix
53      dot_file_path = file_path_without_format + ".dot"
54      png_file_path = file_path_without_format + ".png"
55      pdf_file_path = file_path_without_format + ".pdf"
56
57      print(f"[INFO] Exporting {dot_file_path} ... ", end='', flush=True)
58      export_graphviz(dt, out_file=dot_file_path, class_names=
    class_names_list, feature_names=features_names_list,
59                      rounded=rounded_rectangles, rotate=horizontal,
    proportion=print_percentages, leaves_parallel=parallel_leaves)
60      # For more parameters see: https://scikit-learn.org/stable/modules/
    generated/sklearn.tree.export_graphviz.html
61      print(" [ OK ] ")
62
63      print(f"[INFO] Plotting {file_path_without_format} ... ", end='',
    flush=True)
64      # check_call(['dot', '-Tpng', dot_file_path, '-o', png_file_path]) #
     save as PNG
65      check_call(['dot', '-Tpdf', dot_file_path, '-o', pdf_file_path]) #
    save as PDF
66      print(" [ OK ] ")
67
68 print("[INFO] Loading files and creating the model ... ", end='', flush=
    True)
69
70 # Update the list of CSV files
71 csv_files = glob_get_files_list(input_files_path_labeled_data,
    file_format="csv")
72
73 # Create the splits
```

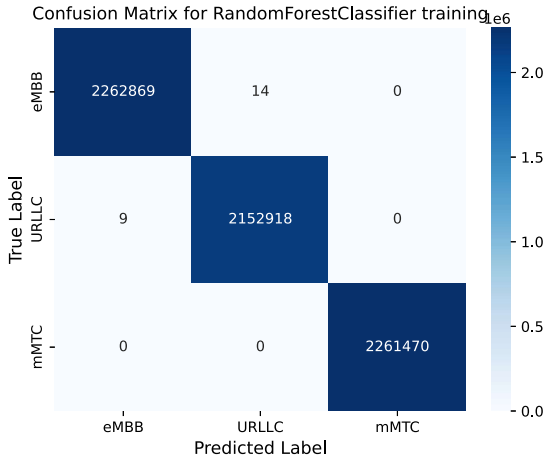
```
74 X_train, X_test, y_train, y_test, features_names_list =
       read_and_split_train_data(csv_files, True, True)
75
76 # Apply SMOTE to oversample the underrepresented class
77 # X_train, y_train = data_oversample(X_train, y_train) # Uncomment to
       apply SMOTE
78
79 # Uncomment one of the lines below
80 # dt = DecisionTreeClassifier() # default parameter DT (run #0)
81 # dt = DecisionTreeClassifier(max_depth=4) # a first parameter that
       could be adjusted is the tree depth (param example for run #0)
82 # dt = DecisionTreeClassifier(min_samples_leaf=1000, min_samples_split
       =10000) # (run #1)
83 # dt = DecisionTreeClassifier(max_depth=6, min_samples_leaf=1000,
       min_samples_split=10000) # (run #2)
84 # dt = DecisionTreeClassifier(max_depth=3) # try another value for the
       tree depth (param example for run #3)
85 # dt = DecisionTreeClassifier(max_depth=2) # try yet another value for
       the tree depth (param example for run #3)
86 # For more parameters see: https://scikit-learn.org/stable/modules/
       generated/sklearn.tree.DecisionTreeClassifier.html
87 dt.fit(X_train, y_train)
88 print(" [ OK ] ")
89
90
91 # Plot "normal" tree (default parameters + rounded boxes)
92 create_and_plot_tree_visualization("vertical_raw", True, False, False)
93 # Plot "normal" tree paralell leaves
94 create_and_plot_tree_visualization("vertical_leaves", True, False, False
       , True)
95 # Plot "normal" tree with percentages
96 create_and_plot_tree_visualization("vertical_percentages", True, True,
       True)
97
98 # Plot horizontal tree with numerical values
99 create_and_plot_tree_visualization("horizontal_raw", True, True, False)
100 # Plot horizontal tree with numerical values
101 create_and_plot_tree_visualization("horizontal_leaves", True, True,
       False, True)
102 # Plot horizontal tree with percentages
103 create_and_plot_tree_visualization("horizontal_percentages", True, True,
        True)
104 print("[INFO] Plotting has finished successfully")
105 end_time = time.time() # record the end of execution
106 print(f"[DEBU] Execution time: {end_time - start_time} s")
```

Source: Created by the author (2025).

APPENDIX R – Test phase confusion matrices

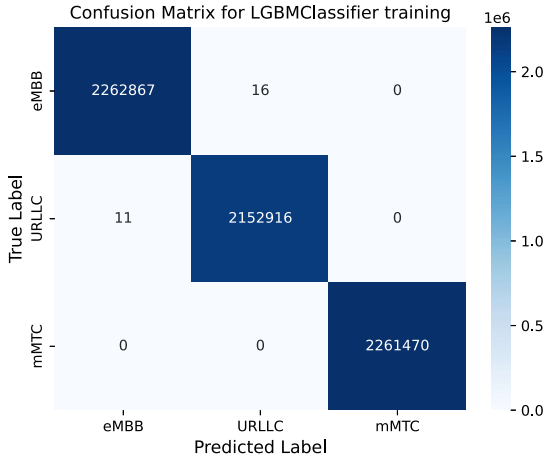
The confusion matrices of the ML test phase are presented in Figures 23–33, illustrating how each model classified the test data on one of the runs. Given that this is a multi-class problem, each matrix comprises four components: TP, FP, TN, and FN). The optimal results (i.e., the TPs) are located along the main diagonal, while the upper triangular portion contains the FPs, the FNs are found in the lower triangular portion, and the TNs are class specific (e.g., the TNs of eMBB class are the other two values from the main diagonal). The performance metrics and further analysis related to these matrices are provided in Subsection *5.2.3.1*, while the raw values of each performance metric detailed in Subsection **2.2.2** is contained in Appendix S. The confusion matrices of all runs are archived on Zenodo (39).

Figure 23 – RF confusion matrix for the test phase



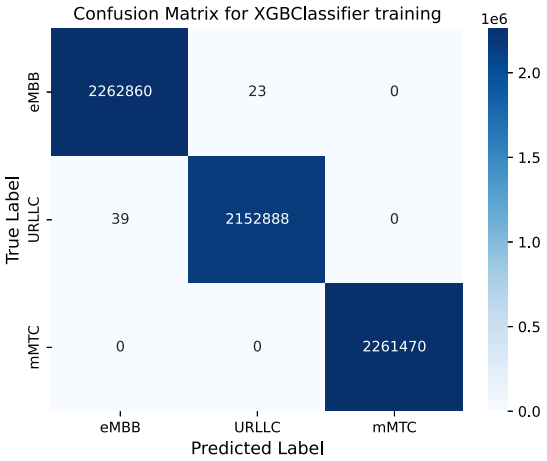
Source: Created by the author (2025).

Figure 24 – LGBM confusion matrix for the test phase



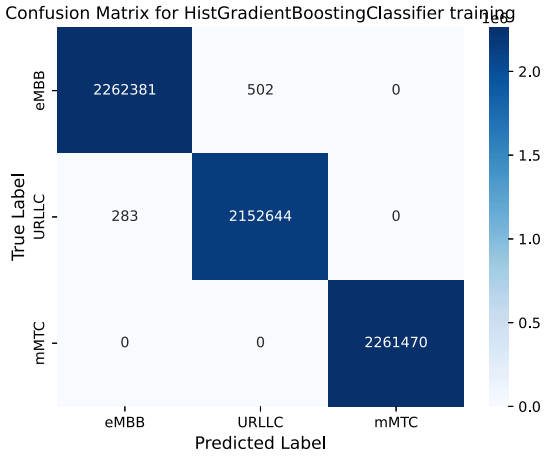
Source: Created by the author (2025).

Figure 25 – XGB confusion matrix for the test phase



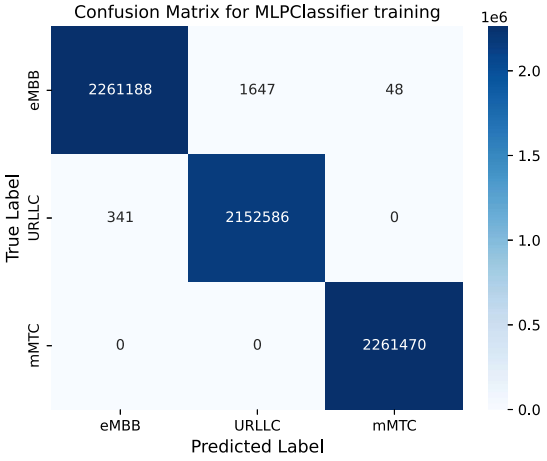
Source: Created by the author (2025).

Figure 26 – HGB confusion matrix for the test phase



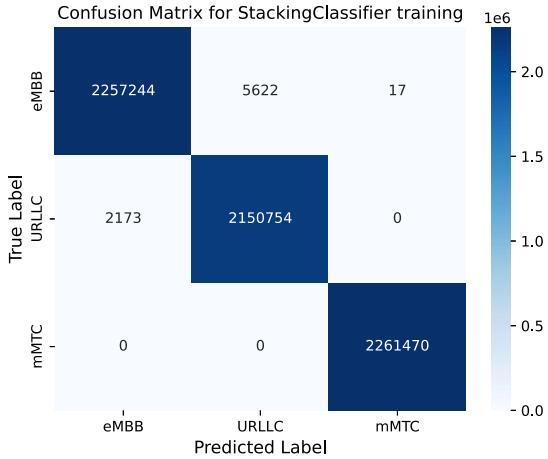
Source: Created by the author (2025).

Figure 27 – MLP confusion matrix for the test phase



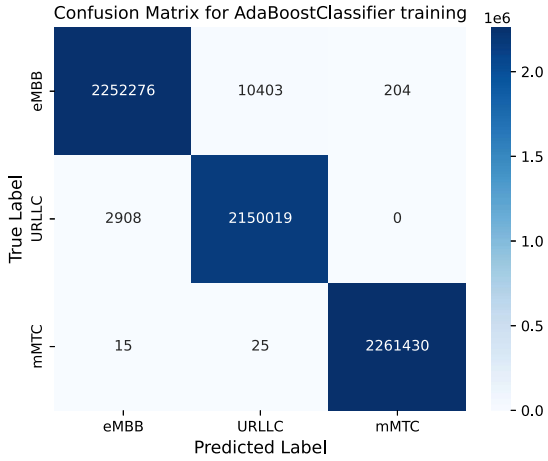
Source: Created by the author (2025).

Figure 28 – Stacking confusion matrix for the test phase



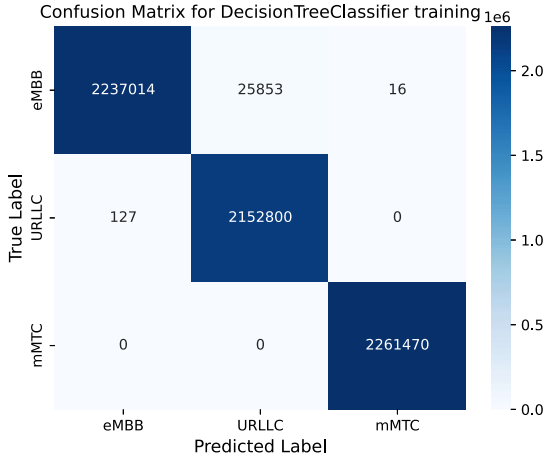
Source: Created by the author (2025).

Figure 29 – AdaBoost confusion matrix for the test phase



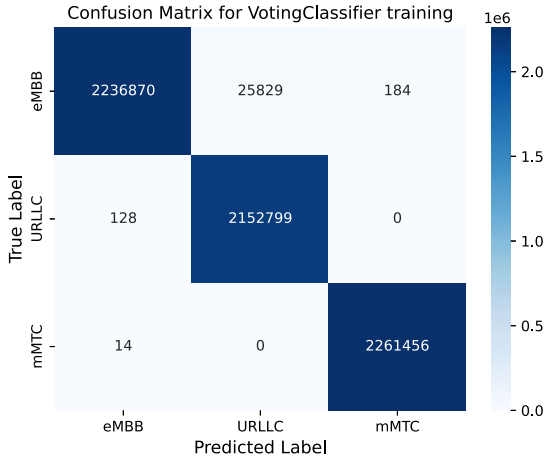
Source: Created by the author (2025).

Figure 30 – DT confusion matrix for the test phase



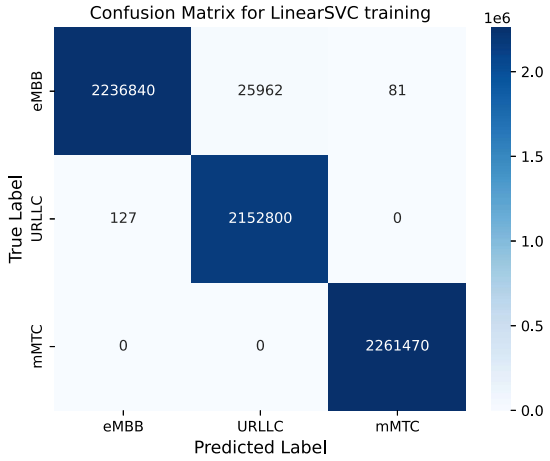
Source: Created by the author (2025).

Figure 31 – Voting confusion matrix for the test phase



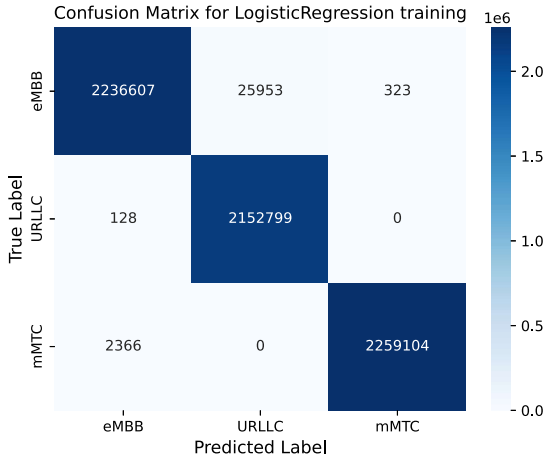
Source: Created by the author (2025).

Figure 32 – Linear SVC confusion matrix for the test phase



Source: Created by the author (2025).

Figure 33 – LR confusion matrix for the test phase



Source: Created by the author (2025).

APPENDIX S – Test phase raw performance metrics

The performance metrics detailed in Subsection **2.2.2** and in Appendix Y (Accuracy, Precision, Recall, F1-score, and AUC score) of the ML test phase are presented in Tables 22–32, illustrating the classification performance of each model related to the test data. Notably, the "average" metrics concern the average values of the three classes and the values are an average of three runs with the fixed `random_state` seed disabled. The analysis related to these tables is provided in Subsection *5.2.3.1*. Finally, the raw model performance results detailed below are archived on Zenodo (39).

Table 22 – Training RF performance results

Performance Metric	**Value**
Accuracy	0.999995756755246
Average Precision	0.999995756758762
Average Recall	0.999995756755246
Average f1-score	0.999995756755408
Class 0 f1-score	0.999993739715435
Class 1 f1-score	0.99999341629356
Class 2 f1-score	1.0
Average AUC Score	0.999999844878934

Source: Created by the author (2025).

Table 23 – Training LGBM performance results

Performance Metric	**Value**
Accuracy	0.99999540731156
Average Precision	0.99999540731582
Average Recall	0.99999540731156
Average f1-score	0.999995407311511
Class 0 f1-score	0.999993371391405
Class 1 f1-score	0.99999287478899
Class 2 f1-score	0.9999998526929
Average AUC Score	0.999999999769697

Source: Created by the author (2025).

Table 24 – Training XGB performance results

Performance Metric	**Value**
Accuracy	0.999988418437847
Average Precision	0.999988418446283
Average Recall	0.999988418437847
Average f1-score	0.999988418437006
Class 0 f1-score	0.999982986655971
Class 1 f1-score	0.999982031579259
Class 2 f1-score	0.999999926307082
Average AUC Score	0.999999999438396

Source: Created by the author (2025).

Table 25 – Training HGB performance results

Performance Metric	**Value**
Accuracy	0.999887379292167
Average Precision	0.9998873820785
Average Recall	0.999887379292167
Average f1-score	0.999887379409384
Class 0 f1-score	0.999833982847908
Class 1 f1-score	0.999825293972727
Class 2 f1-score	0.9999998526929
Average AUC Score	0.999999960191564

Source: Created by the author (2025).

Table 26 – Training MLP performance results

Performance Metric	**Value**
Accuracy	0.999650007188556
Average Precision	0.999650153074434
Average Recall	0.999650007188556
Average f1-score	0.999650007319171
Class 0 f1-score	0.999483462991802
Class 1 f1-score	0.999467328645237
Class 2 f1-score	0.999990349680517
Average AUC Score	0.999998593566159

Source: Created by the author (2025).

Table 27 – Training Stacking performance results

Performance Metric	**Value**
Accuracy	0.998023995798688
Average Precision	0.998029789944568
Average Recall	0.998023995798688
Average f1-score	0.998024086900127
Class 0 f1-score	0.997078630762018
Class 1 f1-score	0.99694395273203
Class 2 f1-score	0.999996904819142
Average AUC Score	N/A

Source: Created by the author (2025).

Table 28 – Training AdaBoost performance results

Performance Metric	**Value**
Accuracy	0.997935786228325
Average Precision	0.997942234507552
Average Recall	0.997935786228325
Average f1-score	0.997935818257243
Class 0 f1-score	0.996953741847231
Class 1 f1-score	0.996854535993169
Class 2 f1-score	0.99994666516008
Average AUC Score	0.999983901260544

Source: Created by the author (2025).

Table 29 – Training DT performance results

Performance Metric	**Value**
Accuracy	0.996120476201887
Average Precision	0.996165562798257
Average Recall	0.996120476201887
Average f1-score	0.996120892703388
Class 0 f1-score	0.994243709366724
Class 1 f1-score	0.994021669151318
Class 2 f1-score	0.999995285556495
Average AUC Score	0.99986424483371

Source: Created by the author (2025).

Table 30 – Training Voting performance results

Performance Metric	**Value**
Accuracy	0.996100857434964
Average Precision	0.996145789417267
Average Recall	0.996100857434964
Average f1-score	0.996101133335754
Class 0 f1-score	0.994214452601538
Class 1 f1-score	0.994030698143807
Class 2 f1-score	0.999957641465848
Average AUC Score	0.999981979855719

Source: Created by the author (2025).

Table 31 – Training Linear SVC performance results

Performance Metric	**Value**
Accuracy	0.996084932787003
Average Precision	0.996130413788997
Average Recall	0.996084932787003
Average f1-score	0.99608524470447
Class 0 f1-score	0.994190665252682
Class 1 f1-score	0.993995878869977
Class 2 f1-score	0.999967662120633
Average AUC Score	N/A

Source: Created by the author (2025).

Table 32 – Training LR performance results

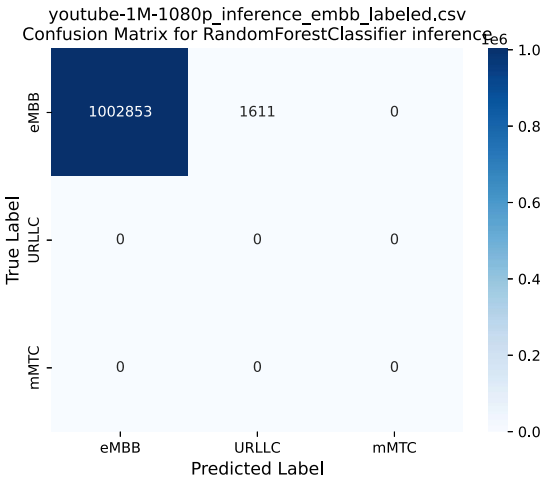
Performance Metric	**Value**
Accuracy	0.995706784798601
Average Precision	0.995748462837908
Average Recall	0.995706784798601
Average f1-score	0.995706963409681
Class 0 f1-score	0.993632665282345
Class 1 f1-score	0.993998326120172
Class 2 f1-score	0.999406996134278
Average AUC Score	0.999612523228833

Source: Created by the author (2025).

APPENDIX T – eMBB inference confusion matrices

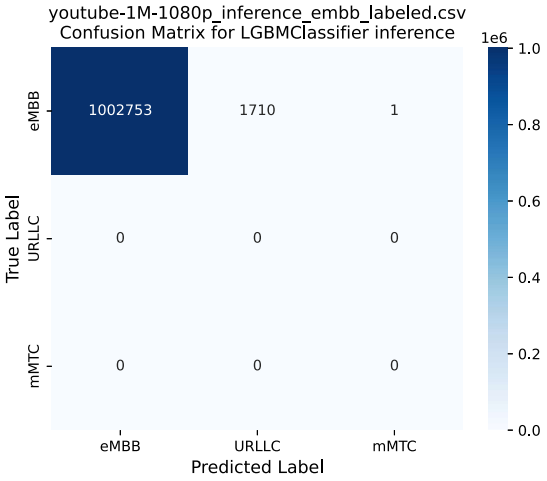
The confusion matrices of the ML inference phase for eMBB class are presented in Figures 34–44, illustrating how each model classified the eMBB inference data. Given that this is a multi-class problem, each matrix comprises four components: TP, FP, TN, and FN). The optimal results (i.e., the TPs) are located along the main diagonal, while the upper triangular portion contains the FPs, the FNs are found in the lower triangular portion, and the TNs are class specific (e.g., the TNs of eMBB class are the other two values from the main diagonal). The performance metrics and further analysis related to these matrices are provided in Subsection *5.2.3.2*. The confusion matrices of all runs are archived on Zenodo (39).

Figure 34 – RF confusion matrix for the inference phase



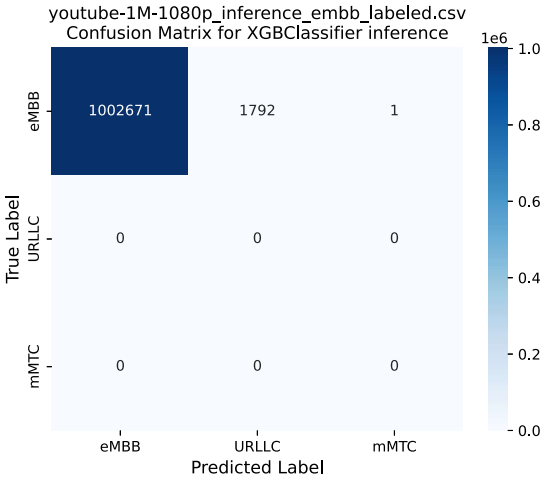
Source: Created by the author (2025).

Figure 35 – LGBM confusion matrix for the inference phase



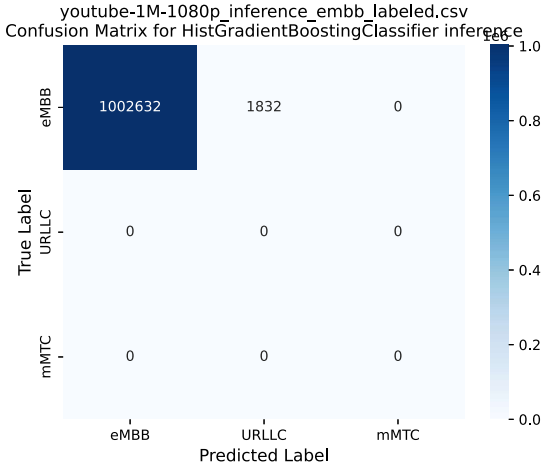
Source: Created by the author (2025).

Figure 36 – XGB confusion matrix for the inference phase



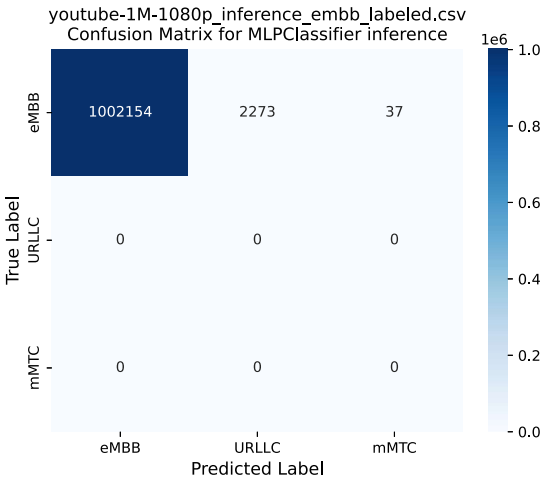
Source: Created by the author (2025).

Figure 37 – HGB confusion matrix for the inference phase



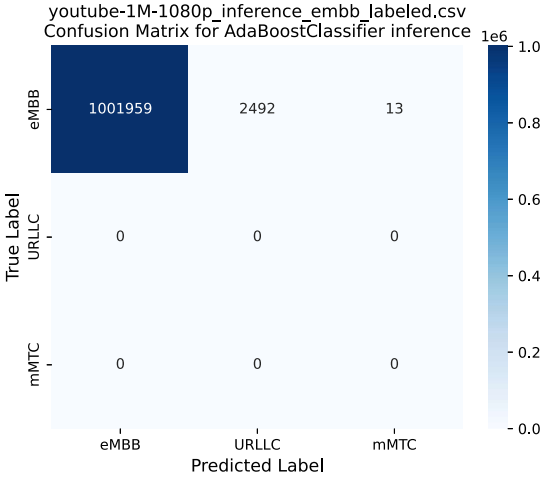
Source: Created by the author (2025).

Figure 38 – MLP confusion matrix for the inference phase



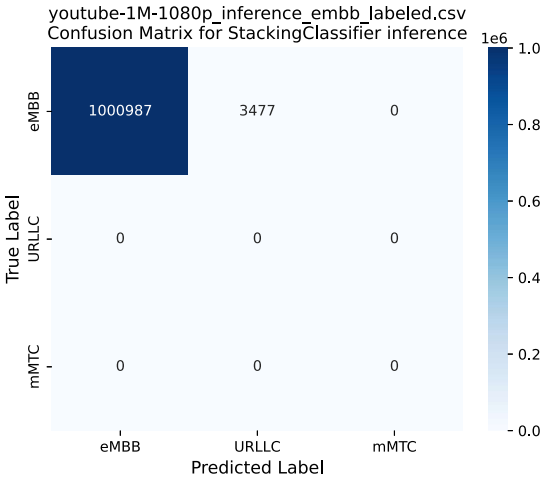
Source: Created by the author (2025).

Figure 39 – AdaBoost confusion matrix for the inference phase



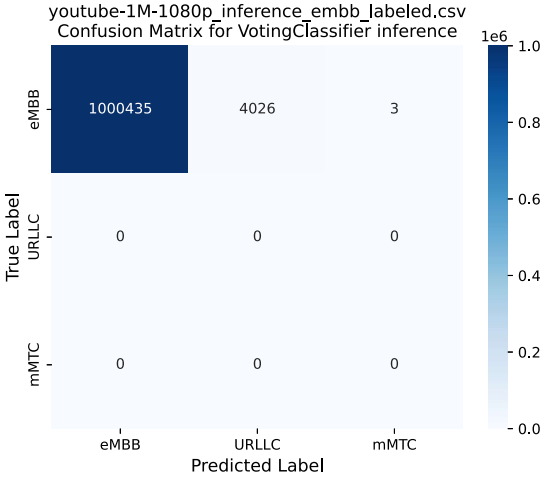
Source: Created by the author (2025).

Figure 40 – Stacking confusion matrix for the inference phase



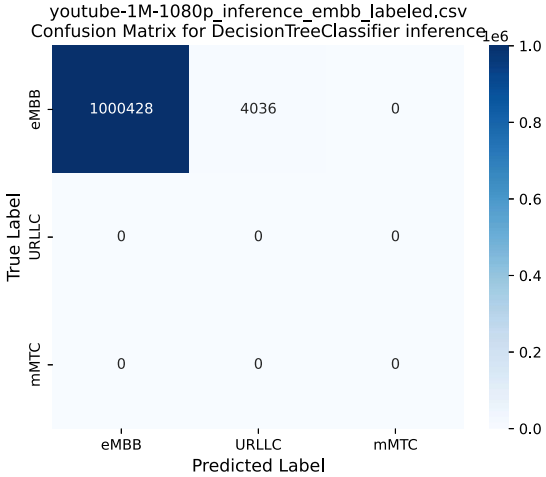
Source: Created by the author (2025).

Figure 41 – Voting confusion matrix for the inference phase



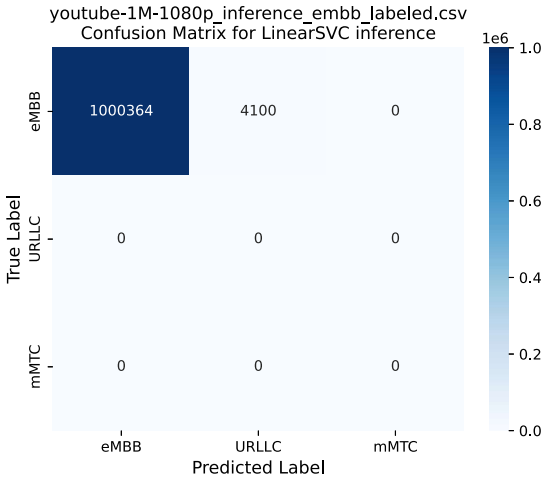
Source: Created by the author (2025).

Figure 42 – DT confusion matrix for the inference phase



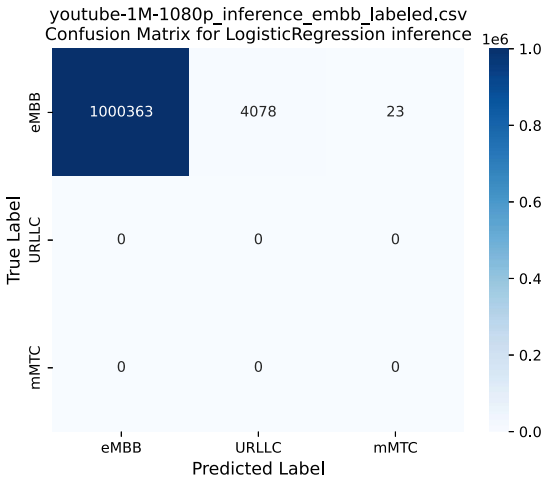
Source: Created by the author (2025).

Figure 43 – Linear SVC confusion matrix for the inference phase



Source: Created by the author (2025).

Figure 44 – LR confusion matrix for the inference phase

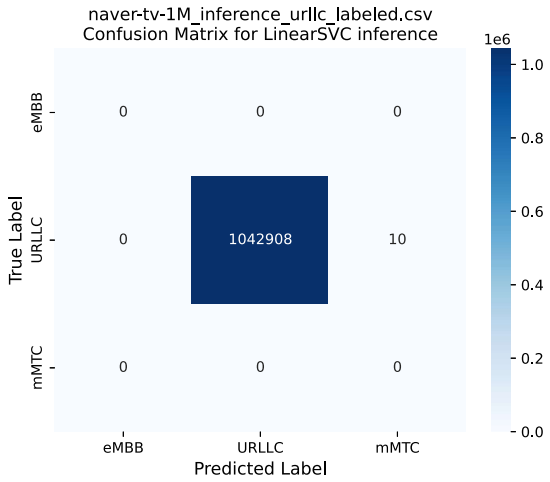


Source: Created by the author (2025).

APPENDIX U – URLLC inference confusion matrices

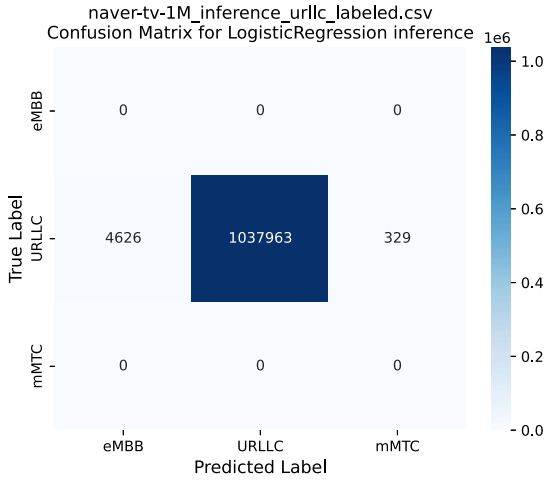
The confusion matrices of the ML inference phase for URLLC class are presented in Figures 45–55, illustrating how each model classified the URLLC inference data. Given that this is a multi-class problem, each matrix comprises four components: TP, FP, TN, and FN). The optimal results (i.e., the TPs) are located along the main diagonal, while the upper triangular portion contains the FPs, the FNs are found in the lower triangular portion, and the TNs are class specific (e.g., the TNs of eMBB class are the other two values from the main diagonal). The performance metrics and further analysis related to these matrices are provided in Subsection *5.2.3.2*. The confusion matrices of all runs are archived on Zenodo (39).

Figure 45 – Linear SVC confusion matrix for the URLLC inference phase



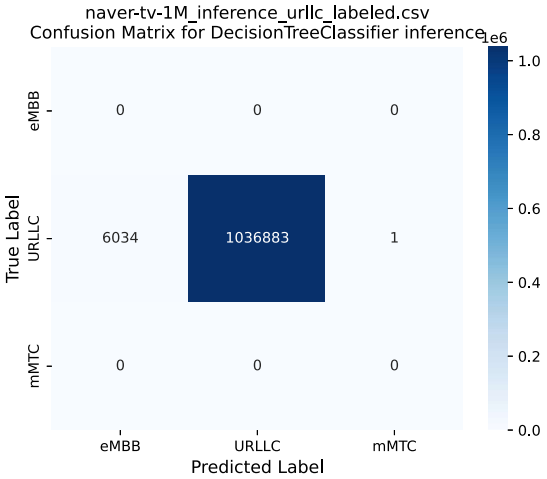
Source: Created by the author (2025).

Figure 46 – LR confusion matrix for the URLLC inference phase



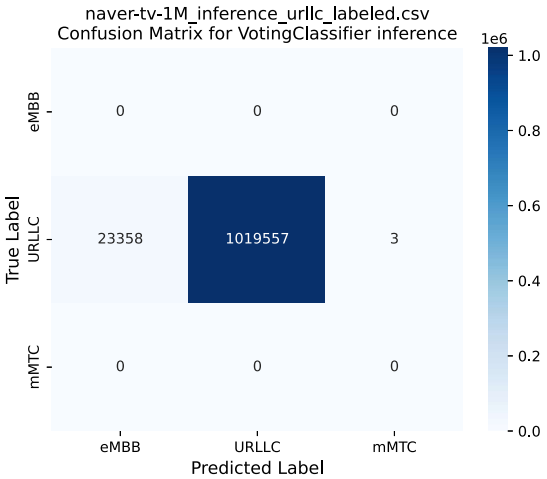
Source: Created by the author (2025).

Figure 47 – DT confusion matrix for the URLLC inference phase



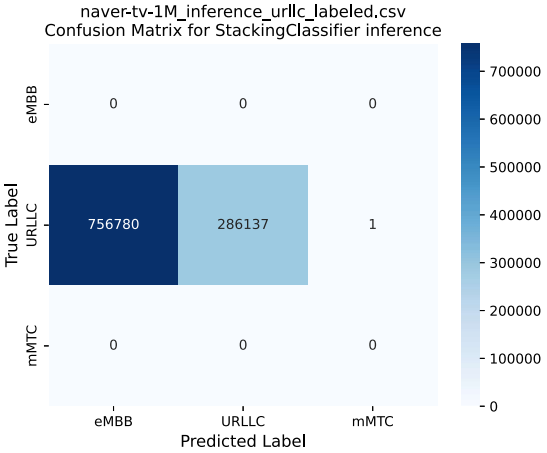
Source: Created by the author (2025).

Figure 48 – Voting confusion matrix for the URLLC inference phase



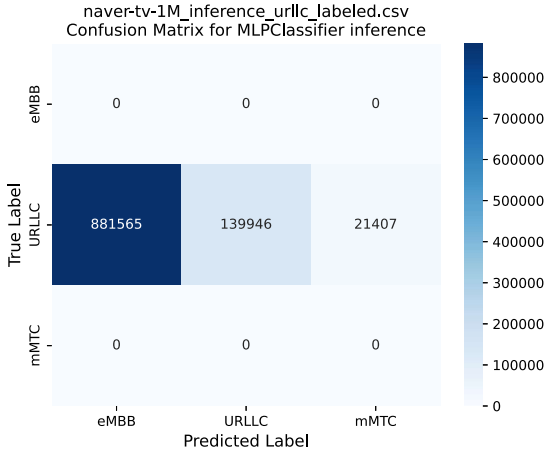
Source: Created by the author (2025).

Figure 49 – Stacking confusion matrix for the URLLC inference phase



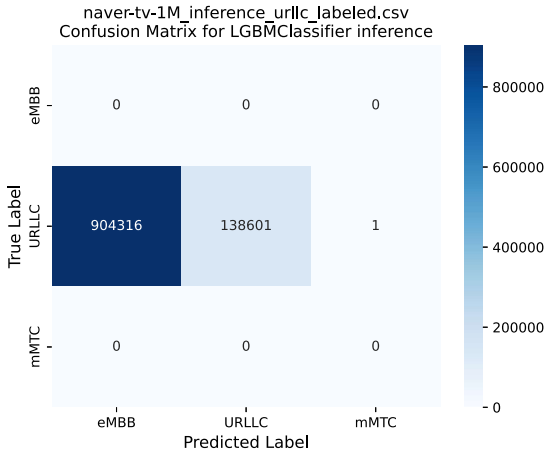
Source: Created by the author (2025).

Figure 50 – MLP confusion matrix for the URLLC inference phase



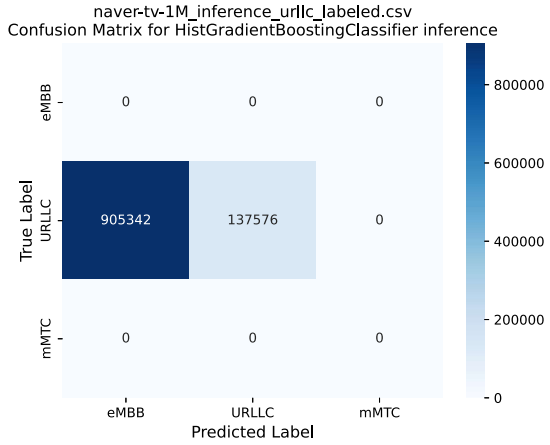
Source: Created by the author (2025).

Figure 51 – LGBM confusion matrix for the URLLC inference phase



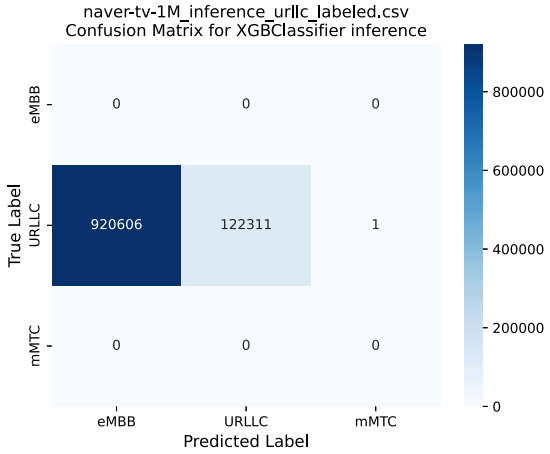
Source: Created by the author (2025).

Figure 52 – HGB confusion matrix for the URLLC inference phase



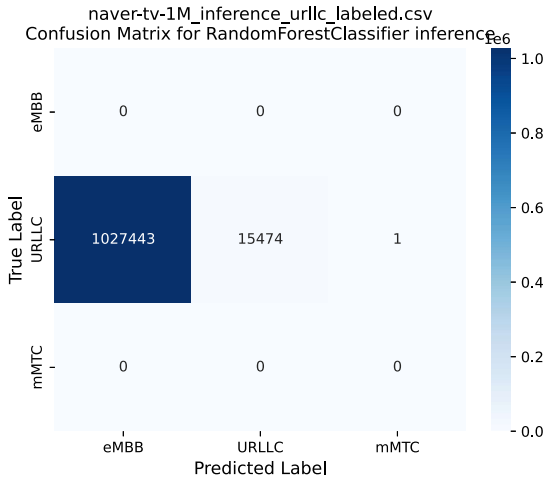
Source: Created by the author (2025).

Figure 53 – XGB confusion matrix for the URLLC inference phase



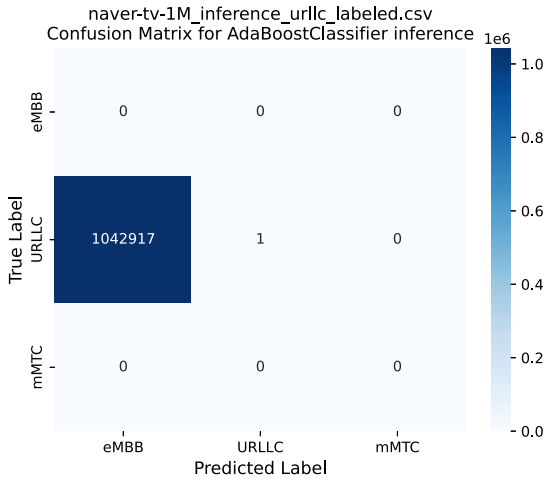
Source: Created by the author (2025).

Figure 54 – RF confusion matrix for the URLLC inference phase



Source: Created by the author (2025).

Figure 55 – AdaBoost confusion matrix for the URLLC inference phase

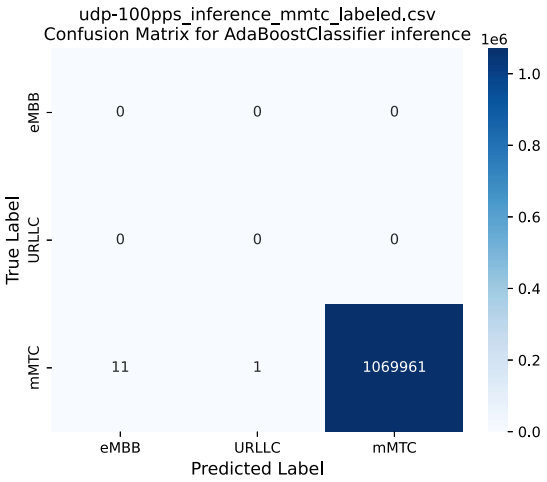


Source: Created by the author (2025).

APPENDIX V – mMTC burst inference confusion matrices

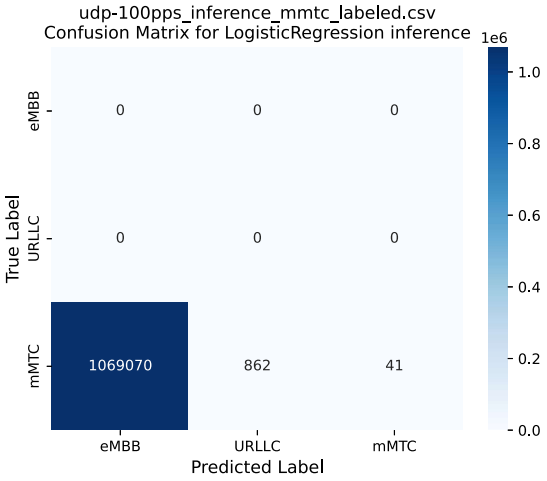
The confusion matrices of the ML inference phase for mMTC class (burst traffic) are presented in Figures 56–66, illustrating how each model classified the mMTC inference data. Given that this is a multi-class problem, each matrix comprises four components: TP, FP, TN, and FN). The optimal results (i.e., the TPs) are located along the main diagonal, while the upper triangular portion contains the FPs, the FNs are found in the lower triangular portion, and the TNs are class specific (e.g., the TNs of eMBB class are the other two values from the main diagonal). The performance metrics and further analysis related to these matrices are provided in Subsection *5.2.3.2*. The confusion matrices of all runs are archived on Zenodo (39).

Figure 56 – AdaBoost confusion matrix for the mMTC burst inference phase



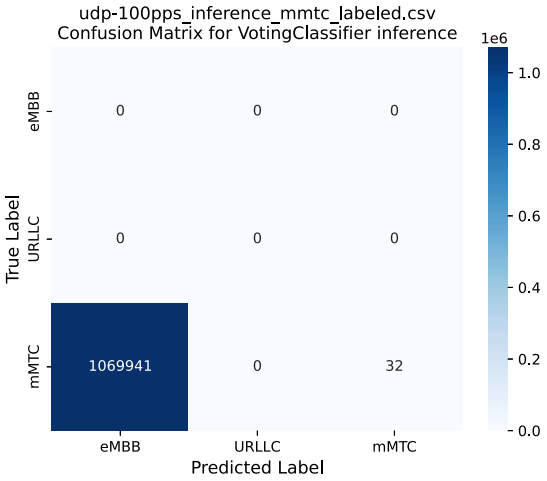
Source: Created by the author (2025).

Figure 57 – LR confusion matrix for the mMTC burst inference phase



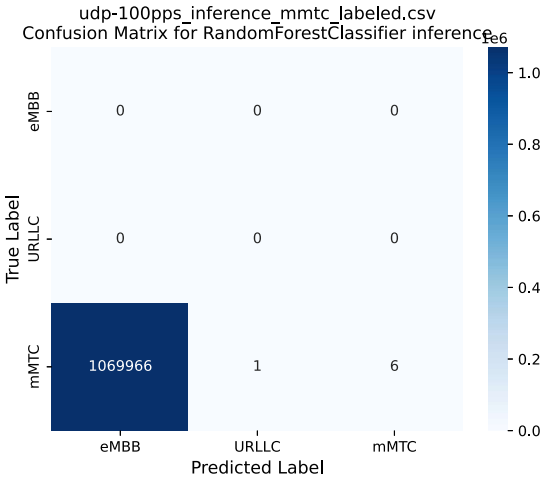
Source: Created by the author (2025).

Figure 58 – Voting confusion matrix for the mMTC burst inference phase



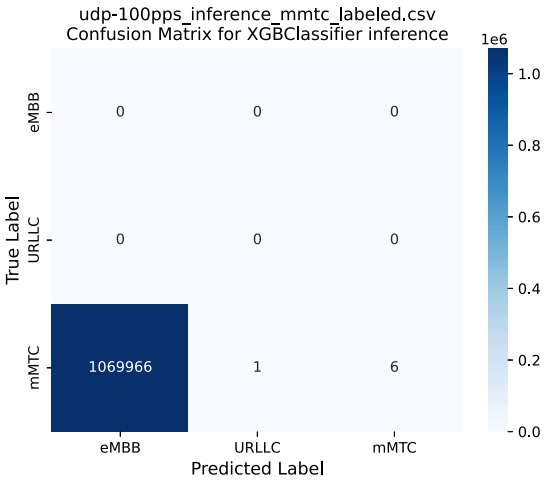
Source: Created by the author (2025).

Figure 59 – RF confusion matrix for the mMTC burst inference phase



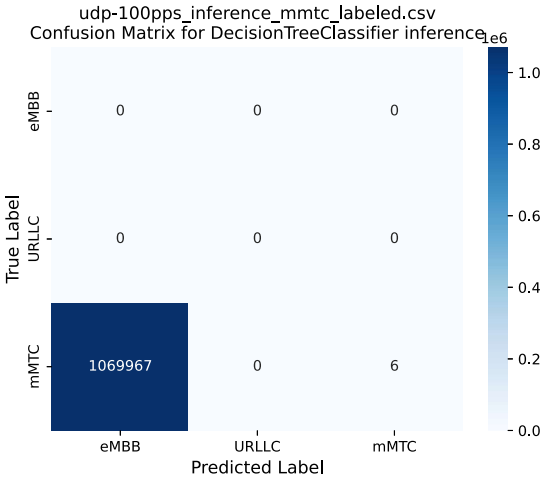
Source: Created by the author (2025).

Figure 60 – XGB confusion matrix for the mMTC burst inference phase



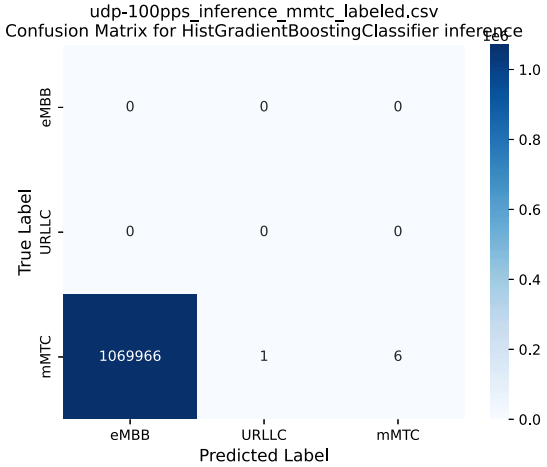
Source: Created by the author (2025).

Figure 61 – DT confusion matrix for the mMTC burst inference phase



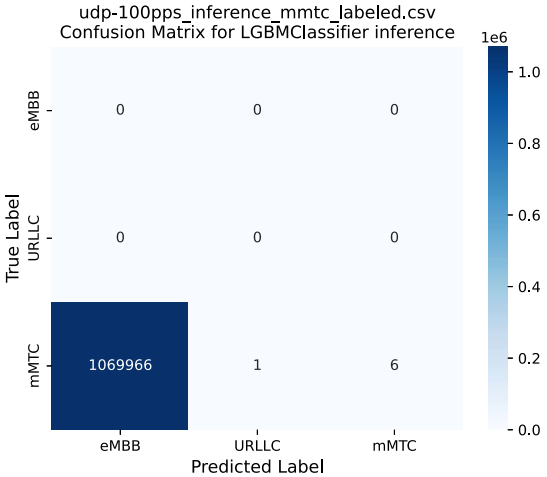
Source: Created by the author (2025).

Figure 62 – HGB confusion matrix for the mMTC burst inference phase



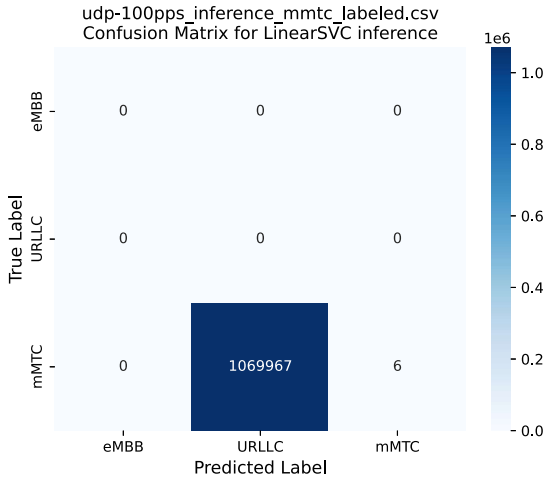
Source: Created by the author (2025).

Figure 63 – LGBM confusion matrix for the mMTC burst inference phase



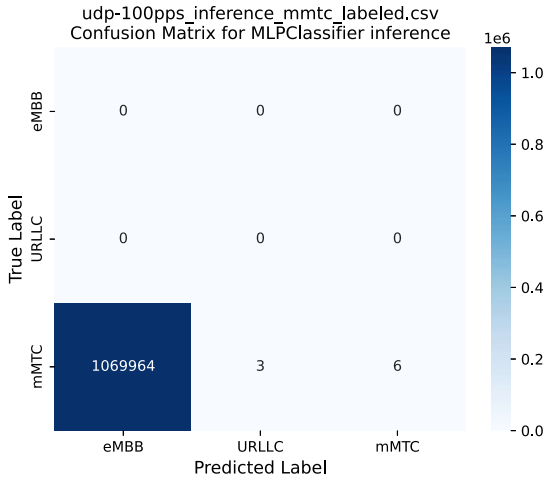
Source: Created by the author (2025).

Figure 64 – Linear SVC confusion matrix for the mMTC burst inference phase



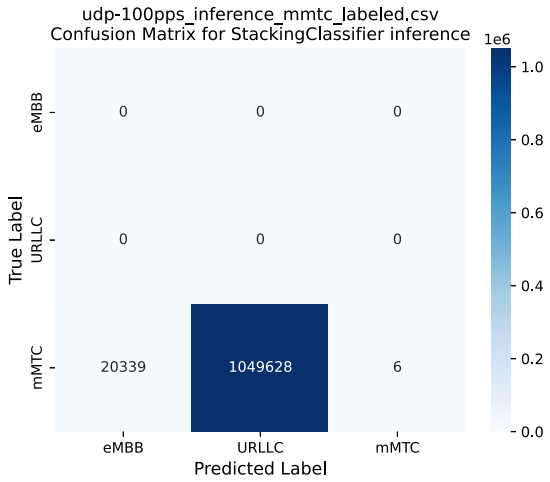
Source: Created by the author (2025).

Figure 65 – MLP confusion matrix for the mMTC burst inference phase



Source: Created by the author (2025).

Figure 66 – Stacking confusion matrix for the mMTC burst inference phase

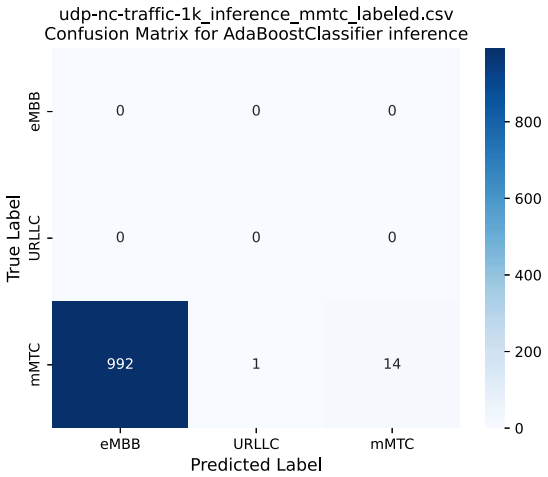


Source: Created by the author (2025).

APPENDIX W – mMTC probabilistic inference confusion matrices

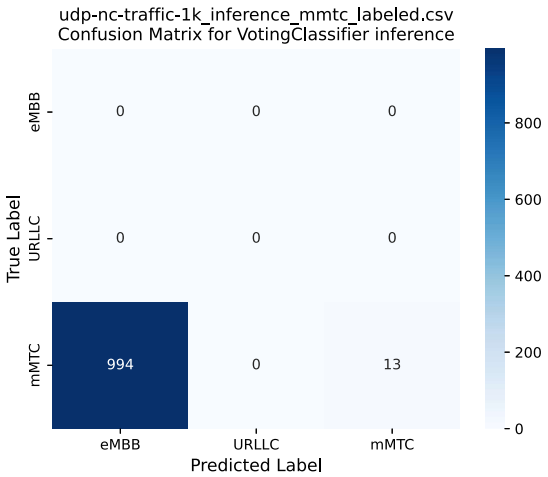
The confusion matrices of the ML inference phase for mMTC class (probabilistic traffic) are presented in Figures 67–77, illustrating how each model classified the mMTC inference data. Given that this is a multi-class problem, each matrix comprises four components: TP, FP, TN, and FN). The optimal results (i.e., the TPs) are located along the main diagonal, while the upper triangular portion contains the FPs, the FNs are found in the lower triangular portion, and the TNs are class specific (e.g., the TNs of eMBB class are the other two values from the main diagonal). The performance metrics and further analysis related to these matrices are provided in Subsection *5.2.3.2*. The confusion matrices of all runs are archived on Zenodo (39).

Figure 67 – AdaBoost confusion matrix for the mMTC probabilistic inference phase



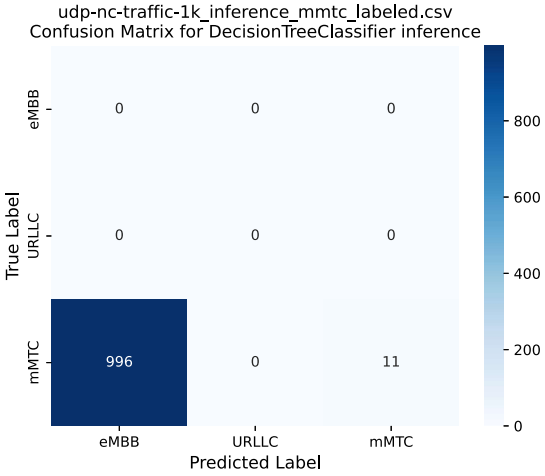
Source: Created by the author (2025).

Figure 68 – Voting confusion matrix for the mMTC probabilistic inference phase



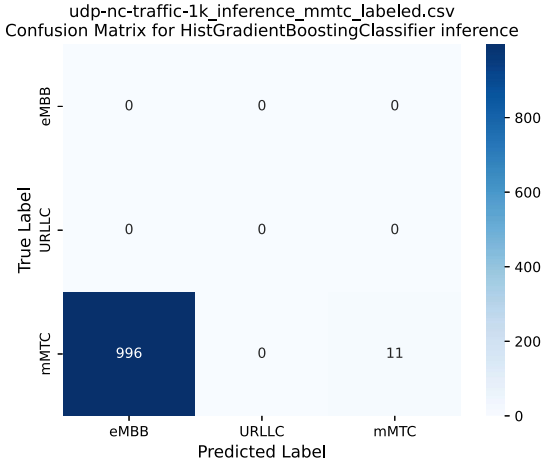
Source: Created by the author (2025).

Figure 69 – DT confusion matrix for the mMTC probabilistic inference phase



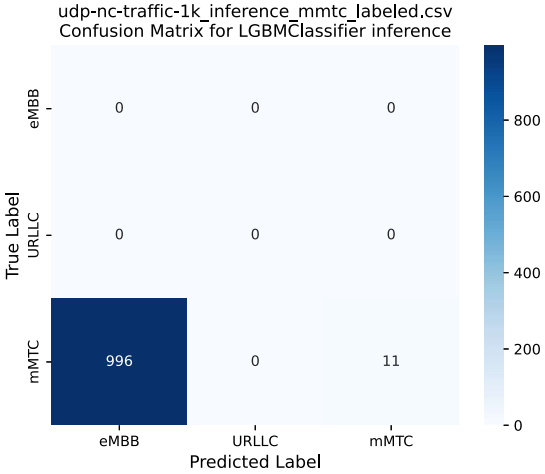
Source: Created by the author (2025).

Figure 70 – HGB confusion matrix for the mMTC probabilistic inference phase



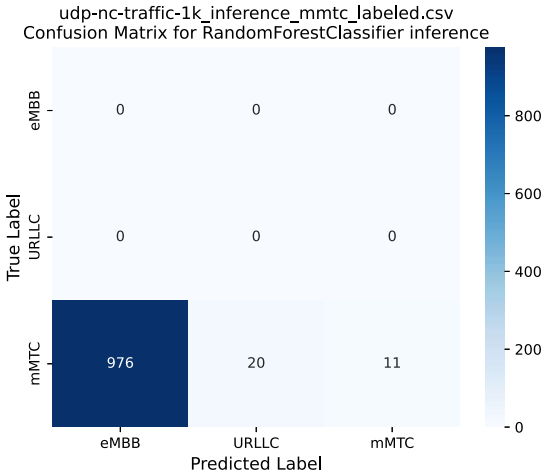
Source: Created by the author (2025).

Figure 71 – LGBM confusion matrix for the mMTC probabilistic inference phase



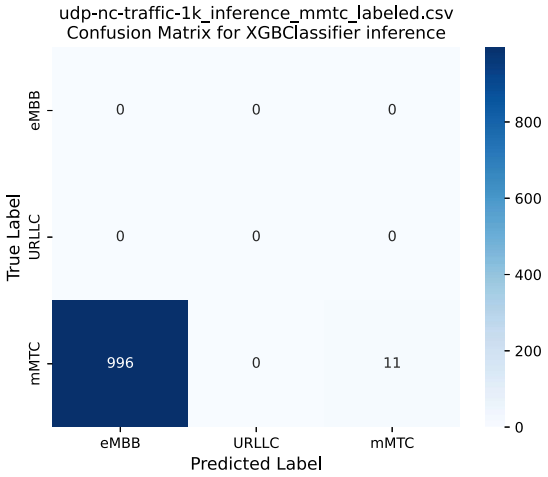
Source: Created by the author (2025).

Figure 72 – RF confusion matrix for the mMTC probabilistic inference phase



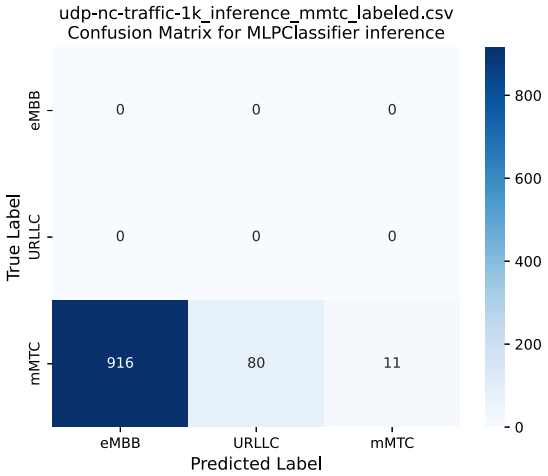
Source: Created by the author (2025).

Figure 73 – XGB confusion matrix for the mMTC probabilistic inference phase



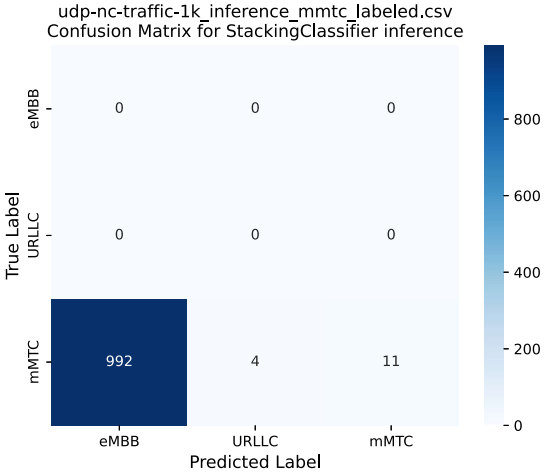
Source: Created by the author (2025).

Figure 74 – MLP confusion matrix for the mMTC probabilistic inference phase



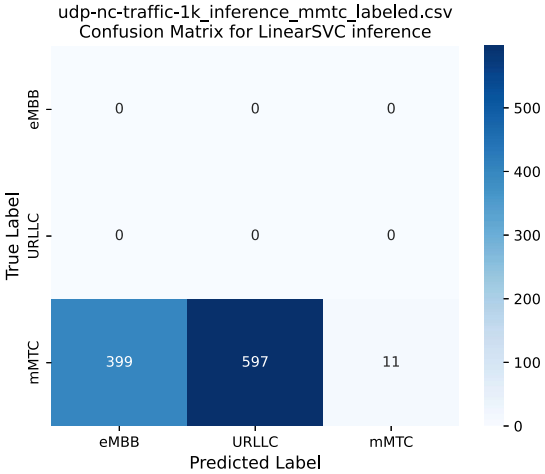
Source: Created by the author (2025).

Figure 75 – Stacking confusion matrix for the mMTC probabilistic inference phase



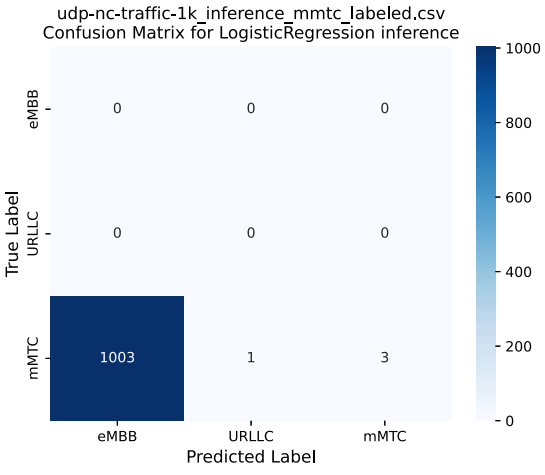
Source: Created by the author (2025).

Figure 76 – Linear SVC confusion matrix for the mMTC probabilistic inference phase



Source: Created by the author (2025).

Figure 77 – LR confusion matrix for the mMTC probabilistic inference phase



Source: Created by the author (2025).

APPENDIX X – Weighted recall definition and example

In multi-class classification problems such as the UE classification performed in this work, it is essential to evaluate the performance of the model not only based on overall accuracy but also on how well it predicts each class. Besides the metrics presented in Subsection **2.2.2**, an additional useful metric for this purpose is the weighted recall.

As shown in Subsection **2.2.2**, recall is calculated by dividing the TPs by the sum of the TPs and FNs. This computation is calculated for each class independently and does not consider the number of instances in each class. While recall provides valuable insight into how well a model identifies positive instances for a specific class, it may not accurately represent the model's performance across all classes, especially in imbalanced datasets (146). Therefore, weighted recall addresses this limitation by incorporating the class distribution, making it a more comprehensive metric for evaluating multi-class classification models.

The weighted recall takes into account the number of instances for each class when calculating the overall recall and is defined as (147, 148):

$$\text{weighted recall} = \frac{\sum_{i=1}^{N_C} \left(\frac{TP_i}{TP_i + FN_i} \times N_i \right)}{N}$$

Where:

- N_C = Total number of classes
- TP_i = True Positives for class i
- FN_i = False Negatives for class i
- N_i = Number of instances for class i (i.e., the sum of true instances for that class)
- N = Total number of instances across all classes (i.e., the sum of all entries in the confusion matrix)

Notably, the term $\frac{TP_i}{TP_i + FN_i}$ is equivalent to Equation 2.3, calculating the recall for class i, which measures the proportion of actual positive instances that were correctly identified by the model. By multiplying this recall by N_i, the recall is weighted according to the prevalence of that i^{th} class in the dataset.

Finally, the sum of these weighted recalls is divided by the total number of instances N to provide an overall measure of recall that reflects the performance across all classes. This approach ensures that classes with more instances have a greater influence on the overall metric, leading to a more balanced evaluation of the model's performance.

As the weighted recall was used in the implementation of the `NWDAF_ml` module (41), it is possible to create a script to validate the execution of the Python libraries (73) and computation of numerical values representing the performance metrics. As the results of most of the eleven models tested converged to a value (as shown in Table 16), only DT and MLP were selected to revalidation in the example implemented in the `accuracy_recall_same_value_proof.py` script listed below.

```
import numpy as np
import pickle

from sklearn import metrics
from util import read_csv

# File paths
working_folder = "./pcap/output/4-ML/"
models_folder = working_folder + "models/" # read the models from here
output_files_path_labeled_data = working_folder + "preprocess/
    labeled_data/" # read preprocessed data from here

models_file_list = [models_folder + "DecisionTreeClassifier.pkl",
    models_folder + "MLPClassifier.pkl"] # Use DT and MLP models
file_name = "udp-nc-traffic-1k_inference_mmtc_labeled.csv" # Using the
    mMTC probabilistic inference data in this example

# Load and prepare splits from labeled training data
df = read_csv(output_files_path_labeled_data + file_name)

# Prepare data for supervised learning
X = df.drop('label', axis=1)  # Features
y_true = df['label']  # Target variable

# Load models from disk
dt_model = pickle.load(open(models_file_list[0], 'rb'))
mlp_model = pickle.load(open(models_file_list[1], 'rb'))

# Classify data
y_pred_dt = dt_model.predict(X)
y_pred_mlp = mlp_model.predict(X)

#########################################################
# Calculating using the Python/Scikit Learn libraries #
#########################################################

# Confusion matrices
conf_matrix_dt = metrics.confusion_matrix(y_true, y_pred_dt, labels
    =[0,1,2])
conf_matrix_mlp = metrics.confusion_matrix(y_true, y_pred_mlp, labels
```

```
    =[0,1,2])

# Print matrices
print("=> Confusion matrices")
print("Decision Tree Confusion Matrix")
print(conf_matrix_dt)
print("Multilayer Perceptron Confusion Matrix")
print(conf_matrix_mlp)

# Print the performance metrics
print("=> Results using libraries")
print(metrics.classification_report(y_true, y_pred_dt, digits=8,
    zero_division=0.0, labels=[0,1,2]))
print(metrics.classification_report(y_true, y_pred_mlp, digits=8,
    zero_division=0.0, labels=[0,1,2]))

############################
# Calculating 'manually' #
############################

# Calculate model accuracy
def calculate_accuracy(conf_matrix):
    # True Positives and True Negatives
    TP = np.diag(conf_matrix).sum()  # Sum of diagonal elements (correct
    predictions)
    total = conf_matrix.sum()  # Total instances
    accuracy = TP / total if total > 0 else 0
    return accuracy

# Calculate model recall for a specific class (e.g., class 2 / mMTC)
def calculate_class_recall(conf_matrix, class_index):
    TP = conf_matrix[class_index, class_index]  # True Positives for the
    class
    FN = conf_matrix[:, class_index].sum() - TP  # False Negatives for
   the class
    recall = TP / (TP + FN) if (TP + FN) > 0 else 0
    return recall

# Calculate model weighted recall
def calculate_weighted_recall(conf_matrix):
    total_instances = conf_matrix.sum()  # Total instances
    weighted_recall = 0.0

    for class_index in range(conf_matrix.shape[0]):
        TP = conf_matrix[class_index, class_index]  # True Positives for
    the class
        FN = conf_matrix[:, class_index].sum() - TP  # False Negatives
```

```
    for the class
77          recall = TP / (TP + FN) if (TP + FN) > 0 else 0
78
79          # Count the number of instances for this class
80          class_instances = conf_matrix[:, class_index].sum()
81
82          # Update weighted recall
83          weighted_recall += recall * class_instances
84
85      # Normalize by total instances
86      weighted_recall /= total_instances if total_instances > 0 else 1
87      return weighted_recall
88
89 # Calculate accuracy, recall and weighted recall for Decision Tree
90 accuracy_dt = calculate_accuracy(conf_matrix_dt)
91 recall_dt_class_2 = calculate_class_recall(conf_matrix_dt, 2)  # Recall
    for class 2
92 weighted_recall_dt = calculate_weighted_recall(conf_matrix_dt)
93
94 # Calculate accuracy, recall and weighted recall for MLP
95 accuracy_mlp = calculate_accuracy(conf_matrix_mlp)
96 recall_mlp_class_2 = calculate_class_recall(conf_matrix_mlp, 2)  #
    Recall for class 2
97 weighted_recall_mlp = calculate_weighted_recall(conf_matrix_mlp)
98
99 # Print results
100 print("=> Results without libraries")
101 print(f"DT Accuracy: {accuracy_dt:.8f}")
102 print(f"DT Recall (Class 2): {recall_dt_class_2:.8f}")
103 print(f"DT Weighted Recall: {weighted_recall_dt:.8f}")
104 print(f"MLP Accuracy: {accuracy_mlp:.8f}")
105 print(f"MLP Recall (Class 2): {recall_mlp_class_2:.8f}")
106 print(f"MLP Weighted Recall: {weighted_recall_mlp:.8f}")
```

Source: Created by the author (2025).

The output of the code above is:

```
1 => Confusion matrices
2 Decision Tree Confusion Matrix
3 [[  0   0   0]
4  [  0   0   0]
5  [996   0  11]]
6 Multilayer Perceptron Confusion Matrix
7 [[  0   0   0]
8  [  0   0   0]
9  [916  80  11]]
10 => Results using libraries
11
```

```
              precision    recall  f1-score   support

           0  0.00000000 0.00000000 0.00000000         0
           1  0.00000000 0.00000000 0.00000000         0
           2  1.00000000 0.01092354 0.02161100      1007

    accuracy                      0.01092354      1007
   macro avg  0.33333333 0.00364118 0.00720367      1007
weighted avg  1.00000000 0.01092354 0.02161100      1007

              precision    recall  f1-score   support

           0  0.00000000 0.00000000 0.00000000         0
           1  0.00000000 0.00000000 0.00000000         0
           2  1.00000000 0.01092354 0.02161100      1007

    accuracy                      0.01092354      1007
   macro avg  0.33333333 0.00364118 0.00720367      1007
weighted avg  1.00000000 0.01092354 0.02161100      1007

=> Results without libraries
DT Accuracy: 0.01092354
DT Recall (Class 2): 1.00000000
DT Weighted Recall: 0.01092354
MLP Accuracy: 0.01092354
MLP Recall (Class 2): 1.00000000
MLP Weighted Recall: 0.01092354
```

Source: Created by the author (2025).

Given the implementation of weighted recall, its calculation is equivalent to the accuracy (as defined in Subsection **2.2.2**). As demonstrated in the example above, there is a possibility of these values converging, which can lead to the equal accuracy and recall values observed in Subsection **5.2.3**. The ready-to-use source code of this appendix and its documentation are also available on GitHub (41).

APPENDIX Y – Receiver Operating Characteristic Area Under the Curve

In addition to the model performance metrics defined in Subsection **2.2.2**, the mathematical formulation of the Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) score is presented.

For binary classification, the ROC curve is constructed by plotting the True Positive Rate (TPR) versus the False Positive Rate (FPR) at various threshold settings (76). These rates are defined as in Equation .1.

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN} \tag{.1}$$

In this context, the Area Under the Curve (AUC) is then defined as in Equation .2, which represents the probability that a randomly chosen positive instance ranks higher than a randomly chosen negative instance.

$$\text{AUC} = \int_0^1 \text{TPR}(t)\, dt \tag{.2}$$

In a multiclass setting with K classes, a common approach is to decompose the problem into multiple binary classification tasks. The One-vs-One (OvO) scheme constructs a classifier for each distinct pair of classes (77). The total number of binary classifiers in this scheme is calculated by Equation .3.

$$\binom{K}{2} = \frac{K(K-1)}{2} \tag{.3}$$

The ROC AUC score is computed for each pair of classes (i, j), denoted AUC_{ij}. In the OvO scheme, the first step identifies all unique pair combinations. Scores are computed by treating one element as the positive class and the other as the negative class, then recomputing with reversed roles and taking the mean of both scores (77). Thus, the OvO macro-average ROC AUC is then calculated by taking the average of these pairwise ROC AUC scores as shown on Equation .4.

$$\text{AUC}_{\text{macro}} = \frac{2}{K(K-1)} \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} AUC_{ij}. \tag{.4}$$

This formulation assigns equal weight to each pair of classes, and consequently, every binary comparison contributes equally to the overall multiclass evaluation metric. For example, consider a three-class classification problem ($K = 3$) with classes A, B, and C. The OvO decomposition results in the following pairs: (A, B), (A, C), and (B, C). Suppose that for these pairs the following ROC AUC scores were obtained:

$$\text{AUC}_{AB} = 0.85, \quad \text{AUC}_{AC} = 0.80, \quad \text{AUC}_{BC} = 0.90.$$

The macro-average ROC AUC would then be:

$$\text{AUC}'_{\text{macro}} = \frac{2}{3(2)}\left(0.85 + 0.80 + 0.90\right) = \frac{2}{6} \times 2.55 = \frac{2.55}{3} \approx 0.85.$$

This value represents the aggregated performance of the classifier over all the pairwise groupings of the classes.