Intelligence in Action: AI Driven Networks - NWDAF

Authors:
Eduardo Lopes, João Neto, Jorge Domingues,
Hugo Ribeiro, Rodrigo Abreu

Supervisors:
Prof. Rui Aguiar, Rafael Direito, Rafael Teixeira

Informatics Project 2024/2025

June 2025

Acknowledgements

We would like to express our gratitude to the supervisors who contributed to the development of this project.

We thank **Prof. Rui Aguiar** for his coordination role and for promoting a stimulating research environment that allowed this project to take shape. We also sincerely thank **Rafael Direito** for his close guidance on the network architecture and integration aspects, and **Rafael Teixeira** for his valuable support in the design and implementation of the machine learning components.

Their expertise, availability, and constructive feedback were essential to the progress and quality of this work.

Keywords

MLOps, 5G, NWDAF, AI, API

Abstract

Modern networks have evolved from static infrastructures into dynamic, intelligent, and adaptive ecosystems. In the context of 5G and Beyond 5G, networks must efficiently manage vast volumes of heterogeneous data, accommodate a wide range of services, and meet stringent requirements for reliability, scalability, and low latency. These demands introduce significant operational challenges, including service degradation due to traffic surges, increased latency, and packet loss, all of which can compromise the quality of experience for end users. Moreover, the physical expansion of infrastructure remains costly and time-consuming.

To address these challenges, this work proposes a modular and scalable Machine Learning Operations (MLOps) pipeline designed to integrate Machine Learning (ML) and automation into network operations. The architecture enables real-time data collection, processing, and intelligent decision-making, forming the foundation for self-optimizing, self-healing network functions. By embedding ML models directly into the network workflow, the system enables continuous learning and adaptation to evolving traffic patterns and usage behaviors.

The pipeline is designed to support a wide range of use cases, including performance forecasting, traffic classification, and anomaly detection. To validate the implementation, we apply the pipeline to an anomaly detection scenario, where it is tasked with identifying various network attacks within mixed traffic flows. This use case serves as a proof of concept, demonstrating the system's ability to distinguish between benign and malicious activity using ML models. The successful application of the pipeline in this context highlights its effectiveness in real-world conditions and underscores its potential to enhance network resilience, reduce operational overhead, and ensure consistent quality of service across dynamic and demanding environments.

Contents

1 Introduction

As the global demand for seamless connectivity, high-speed data transmission, and real-time services continues to grow, network infrastructures must evolve to meet increasingly complex performance, scalability, and reliability requirements. These stringent requirements cannot be supported by traditional static network architectures are no longer sufficient to support the dynamic and heterogeneous demands of modern digital ecosystems. In this context, the fifth generation of mobile networks (5G) and Beyond-5G (B5G) networks represent a significant technological leap, introducing capabilities such as ultra-low latency, massive machine-type communications, and enhanced mobile broadband. However, to fully unlock the potential of these next-generation networks, a shift towards intelligent, data-driven management and automation is essential.

The complexity of managing modern networks, ranging from physical and virtual infrastructure to service orchestration, has reached a point where manual configuration and static policies can no longer guarantee optimal performance. To address this, the integration of Machine Learning (ML) techniques and Machine Learning Operations (MLOps) into network architectures is emerging as a critical approach. These technologies enable the network to continuously learn from operational data, predict future states, and adapt in near-real time, ultimately leading to self-managing and self-optimizing networks.

One of the central frameworks supporting this paradigm shift is the Network Data Analytics Function (NWDAF), introduced as part of the 3GPP 5G core (5GC) specification. NWDAF enables the collection, processing, and exposure of analytical insights about network behavior through standardized APIs. These insights can then be used to make data-driven decisions, such as dynamic resource allocation, anomaly detection, load forecasting, and quality of service (QoS) optimization.

This project is situated within this evolving landscape. It is motivated by the urgent need to design a modular and scalable architecture that leverages ML, automation, and robust data pipelines to enhance 5G network operations. The architecture must support end-to-end functionality, from data ingestion and preprocessing to model training and inference, while enabling real-time communication between network components via standardized APIs and streaming middleware. By doing so, the system can enable predictive insights and automation decisions that enhance operational efficiency and network adaptability.

This need is further amplified by the operational challenges faced by network operators, who are under pressure to reduce operational costs, accelerate service delivery, and minimize downtime. Intelligent network automation aligns directly with these objectives. By embedding ML into the network fabric, the system can identify patterns in data that would be difficult or impossible for humans to detect in real time.

Ultimately, the broader motivation is to implement a proof of concept (PoC) of what is envisioned to be an NWDAF. Rather than aiming for a fully autonomous system from the start, the focus is on exploring how such a component can be designed and integrated, while identifying key challenges, limitations, and opportunities for improvement. Through this project, we aim to gain hands-on experience and generate insights that can inform future development efforts in intelligent network management.

The remainder of this report is organized as follows: Section 2 provides the necessary background on 5G network architecture and the principles of MLOps, setting the technical foundation for the proposed work. Section 3 reviews the state of the art, highlighting relevant research and identifying existing gaps. Section 4 introduces the proposed solution, detailing the objectives, use case selection, system requirements, and architectural components. Section 5 presents the results obtained from implementing and validating the system. Section 6 discusses the challenges encountered, design decisions made, and insights gained during development. Section 7 outlines future work and potential improvements to extend the system's capabilities. Finally, Section 8 concludes the report with a summary of the project's contributions and outcomes.

2 Background

2.1 5G Network Context

The 5G represents a fundamental shift from traditional static infrastructures toward intelligent, adaptive, and software-defined architectures. At the heart of this transformation is the 5GC Network (shown in Figure 1), which acts as the central nervous system of the entire system. It manages authentication, mobility, policy enforcement, security, billing, and (most critically for this project) data routing and exposure. Compared to previous generations, the 5GC is designed to be more modular, cloud-native, and service-oriented, enabling it to support both consumer-centric and industrial use cases with high throughput and ultra-low latency demands [1].

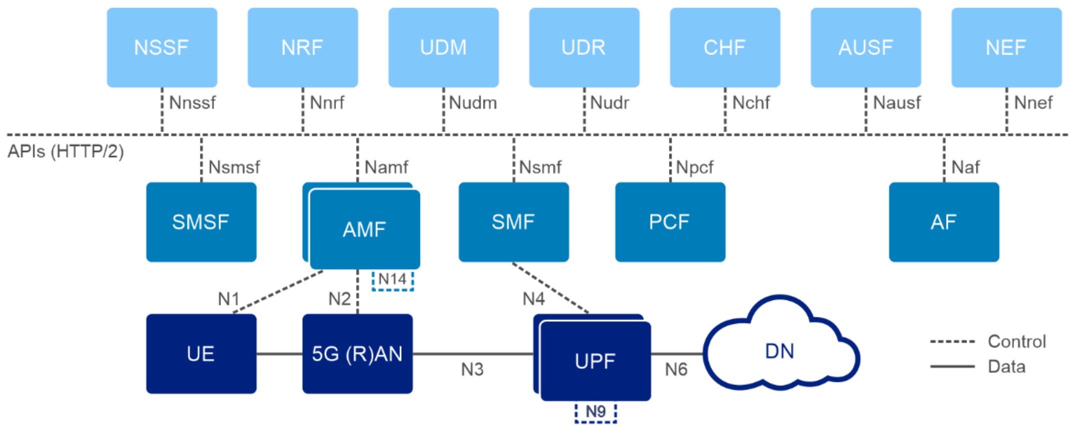


Figure 1: 5G Core Architecture.
Source: Dell Technologies Info Hub (2023), *The 5G Core Network Demystified*. Available at: `https://infohub.delltechnologies.com/en-us/p/the-5g-core-network-demystified/`

The 5GC's architecture is built on a Service-Based Architecture (SBA), in which individual Network Functions (NFs) are decomposed into distinct microservices. These include the Access and Mobility Management Function (AMF) for user registration and mobility handling, and the Session Management Function (SMF), which manages session establishment and communication tunnels between the access network and the User Plane Function (UPF). The UPF is responsible for packet routing and inspection, QoS enforcement, and external data network interconnection. Together, SMF and UPF form the backbone of data path management in the 5GC.

Other key components include the Policy Control Function (PCF) for rule enforcement across slices and services, the Network Repository Function (NRF) for NF discovery and registration, and the Network Exposure Function (NEF), which handles secure exposure of network capabilities and data to external applications. These functions are supported by centralized data services such as the Unified Data Repository (UDR) and Unified Data Management (UDM) for storing and accessing subscriber and policy-related data. The Authentication Server Function (AUSF) manages user authentication, and the Network Slice Selection Function (NSSF) aids in selecting the appropriate network slice for each user or service. The Application Function (AF) provides application-layer capabilities and can influence network behavior by interacting with NEF and PCF.

This highly modular structure enables 5G networks to support advanced features like network slicing, edge computing, and massive Internet of Things (IoT), which are crucial for modern digital services. The core's flexibility allows operators to scale from small private deployments to massive national infrastructures.

In this context, the NWDAF emerges as a pivotal component in the 5GC architecture, introduced by 3GPP to meet the increasing demands for intelligent and automated network management. NWDAF is a standardized function designed to collect, analyze, and expose analytics data (both real-time and historical) to other NFs and external entities. Its primary objective is to enable closed-loop automation by providing actionable insights that drive autonomous decision-making across the network.

Functionally, NWDAF operates as an analytics engine within the SBA of 5G, supporting multiple use cases such as anomaly detection, mobility pattern analysis, user experience prediction, and network slice performance monitoring. It interfaces with various NFs, such as the AMF, SMF, and PCF, to gather network metrics and expose aggregated insights through standardized APIs. This data can be consumed by internal functions or exposed externally to application functions (AFs) and third-party systems via the NEF, enabling intelligent orchestration and service optimization.

NWDAF supports both statistical (batch) analytics and event-driven (streaming) analytics, making it suitable for both long-term trend analysis and real-time operational responses. By leveraging advanced data analytics and potentially ML models, NWDAF enables the network to detect performance anomalies, anticipate failures before they impact service quality, and dynamically adapt to traffic changes. This plays a critical role in supporting predictive maintenance, adaptive QoS provisioning, and the self-optimization capabilities envisioned for Beyond 5G networks.

Ultimately, NWDAF represents a foundational step toward the automation and intelligence goals of 5G, aligning network operations with MLOps principles by integrating data pipelines, model inference, and feedback loops within the core infrastructure. Its introduction reflects the industry's shift toward AI-driven, data-centric networking, positioning it as a key enabler for future network evolution.

2.2 ML Context

As modern networks increase in complexity and scale, the volume of telemetry data and operational metrics they generate grows exponentially. Traditional approaches to network management (primarily rule-based or manually configured) are no longer sufficient to ensure real-time adaptability, reliability, or scalability. In this context, ML becomes a powerful tool for enabling data-driven automation, allowing networks to dynamically respond to changing conditions with minimal human intervention. MLOps has emerged as a discipline that addresses these challenges by applying principles from DevOps and software engineering to the ML domain.

However, leveraging ML in production environments, particularly in critical systems like 5G networks, requires more than just model development. It requires a structured and automated lifecycle management process for ML workflows, known as MLOps. MLOps draws inspiration from DevOps practices in software engineering and extends them to the domain of data science, enabling continuous integration, continuous delivery, observability, and version control of models and data (shown in Figure 2).

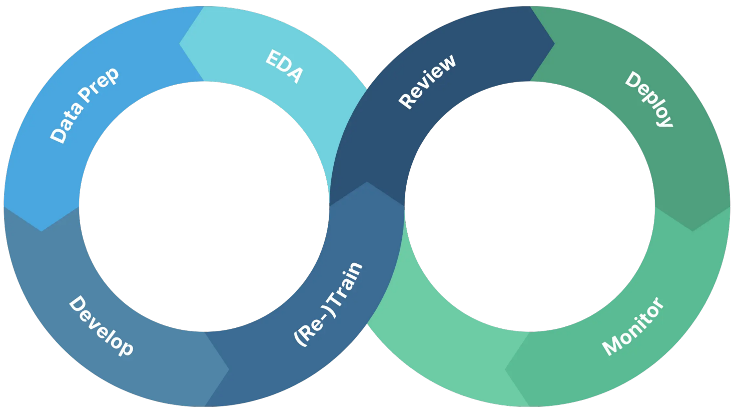


Figure 2: MLops Lifecycle

The ML lifecycle is a continuous, iterative process that spans from initial data preparation to post-deployment monitoring. As shown in Figure 2, it begins with data preparation and exploratory data analysis (EDA), which are essential for understanding and shaping the dataset. This is followed by model development and training, where algorithms are iteratively refined and validated. After review and testing, models are deployed to production environments. Continuous monitoring ensures models remain effective over time, enabling feedback-driven retraining and improvement. This cyclical structure supports adaptability and long-term performance in dynamic, data-driven systems.

An MLOps pipeline is a structured and automated framework that manages the end-to-end lifecycle of ML workflows, from data collection and preprocessing to model training, deployment, monitoring, and retraining. Its goal is to ensure that ML models are delivered in a reproducible, scalable, and maintainable way, while supporting continuous integration (CI), continuous deployment (CD), and continuous training (CT). A mature MLOps pipeline not only accelerates experimentation and iteration cycles but also enforces automation, governance, and traceability, enabling organizations to deploy models into production with confidence and efficiency.

In the context of this project, MLOps is central to the development of an intelligent, modular, and scalable architecture that integrates seamlessly with 5G and B5G network components. The pipeline we designed spans several stages, each of which contributes to the overall goal of enabling predictive and adaptive network behavior.

1. Data Ingestion Data ingestion is the entry point of a machine learning pipeline, particularly in network environments where telemetry and performance metrics are continuously generated. This stage involves the real-time collection of high-volume data streams, ensuring timely and scalable data flow into the system.

2. Data Preprocessing Raw data often contains noise, missing values, and inconsistencies. The preprocessing stage involves cleaning, normalizing, and enriching the data to ensure quality and consistency. It also includes feature extraction, transforming raw inputs into meaningful attributes that can enhance model performance.

3. Model Training and Evaluation This stage focuses on developing predictive models based on the processed data. Typical objectives include detecting anomalies, forecasting failures, or classifying events. The training process incorporates model validation techniques and evaluation through metrics such as precision, recall, and F1-score to ensure effectiveness and generalizability.

4. Model Deployment and Inference Once trained, models are prepared for deployment in a production environment. This involves exposing them through inference services that can handle real-time data and generate predictions or classifications. These outputs are then used to support automated or semi-automated decision-making processes.

5. Continuous Integration/Deployment (CI/CD) CI/CD practices are applied to automate testing, validation, and deployment of model updates. This facilitates rapid iteration while maintaining system reliability and minimizing the risk of disruption when models are replaced or improved.

6. Monitoring and Observability Monitoring is essential for tracking model performance and detecting issues such as data drift or concept drift over time. Observability tools enable the implementation of feedback loops, allowing for ongoing evaluation and retraining of models to adapt to changing conditions and maintain accuracy.

3 State of the Art

In recent years, the integration of ML into 5G network operations has attracted significant attention from both academia and industry. Several works have explored the potential of applying ML to enhance network intelligence, with a particular focus on the NWDAF, a key enabler of data-driven automation in 5G systems. For example, Mekrache et al. [5] propose combining NWDAF with ML techniques to detect abnormal traffic in B5G environments, while Nisha et al. [7] introduce a smart data analytics system that leverages ML algorithms to improve communications in 5G networks. Similarly, Ferreira et al. [3] present a demonstration system to improve network performance by analyzing function and slice loads.

Although these contributions highlight promising ML-based capabilities for network optimization and anomaly detection, they do not address the broader lifecycle challenges associated with ML in production systems. In particular, none of the referenced works explores CI/CD of ML models or strategies for automatic retraining in response to evolving data patterns. This represents a critical gap, as real-world deployments require robust MLOps practices to ensure model accuracy and relevance over time. The absence of mechanisms to detect and mitigate model drift (where the data distribution changes and affects model performance) further limits the practical applicability of these solutions.

Another key shortcoming across [3], [5], and [7] is the lack of end-to-end architectural considerations that demonstrate how ML can be operationalized within NWDAF-like systems. While theoretical models and simulations are well-explored, there remains a gap in proof-of-concept implementations that bridge the divide between research and real-world deployment of AI-driven NFs.

In terms of feature extraction and data preparation for ML, Sarhan et al. [8] highlight the importance of standardized and informative network features for intrusion detection. Their work points to nProbe as an efficient and widely adopted flow exporter capable of generating rich NetFlow/IPFIX features, which are suitable for ML-based traffic analysis. Despite its potential, existing research does not incorporate nProbe into a fully operational ML pipeline integrated with NWDAF architectures, leaving open questions about how such tools can be leveraged in a modular and scalable way.

In summary, while recent literature demonstrates the value of integrating ML into 5G analytics, it often stops short of addressing the deployment, lifecycle management, and modularization challenges required for real-world applications. This gap motivates the design of architectures that operationalize MLOps within NWDAF-compatible systems using technologies like nProbe for data enrichment and Kafka for real-time event streaming.

4 Proposed Solution

4.1 Objectives & Expected Results

The primary objective of this project is to design and implement a scalable and modular data analytics architecture tailored for 5G networks, with a strong focus on integrating ML and MLOps practices into the core of network intelligence. This architecture aims to collect and analyze data exposed by 5GC components, enabling data-driven insights, anomaly detection, and automation of network decision-making processes. Central to this effort is the development of a PoC system that demonstrates the feasibility of using machine learning to detect anomalies and generate insights from 5GC data. The PoC will validate the system's ability to process real or simulated data in near real time, trigger relevant alerts or actions based on ML outputs, and provide meaningful visualizations to operators. This measurable outcome will help assess how ML-driven analytics can contribute to better network performance, resource optimization, and service reliability.

To achieve these goals, the project aims to deliver several concrete outcomes. First, a collection of representative use cases will be defined, highlighting how ML-based intelligence can be leveraged to identify performance bottlenecks, detect anomalies, and support real-time adaptive optimizations. These use cases will guide the architectural design and system requirements.

Second, a robust data analytics pipeline will be implemented, ensuring seamless communication between its services (from raw data collection and processing to model training and inference). The architecture will include integration with key 5G components, ensuring compatibility with the SBA and compliance with 3GPP standards, particularly in the context of NWDAF-like behavior.

Third, the project will involve the development and integration of ML models capable of processing network data to support classification tasks, such as anomaly detection. These models will be deployed through an end-to-end MLOps pipeline, supporting retraining and continuous delivery of intelligence to the network in a reliable and automated manner.

Finally, the architecture will include mechanisms for deployment, monitoring, and visualization, providing a user-friendly interface for network operators to interact with the system and gain insights into the network's performance and behavior. Visualization tools will support real-time observability of inference results.

4.2 Actors

The target clients for this software are telecommunications companies, particularly those offering 5G network services and seeking to enhance the efficiency and self-optimization of their infrastructure. The primary objective is to provide a system that enables networks to autonomously optimize their performance, minimizing the need for manual intervention while improving overall service quality and user experience.

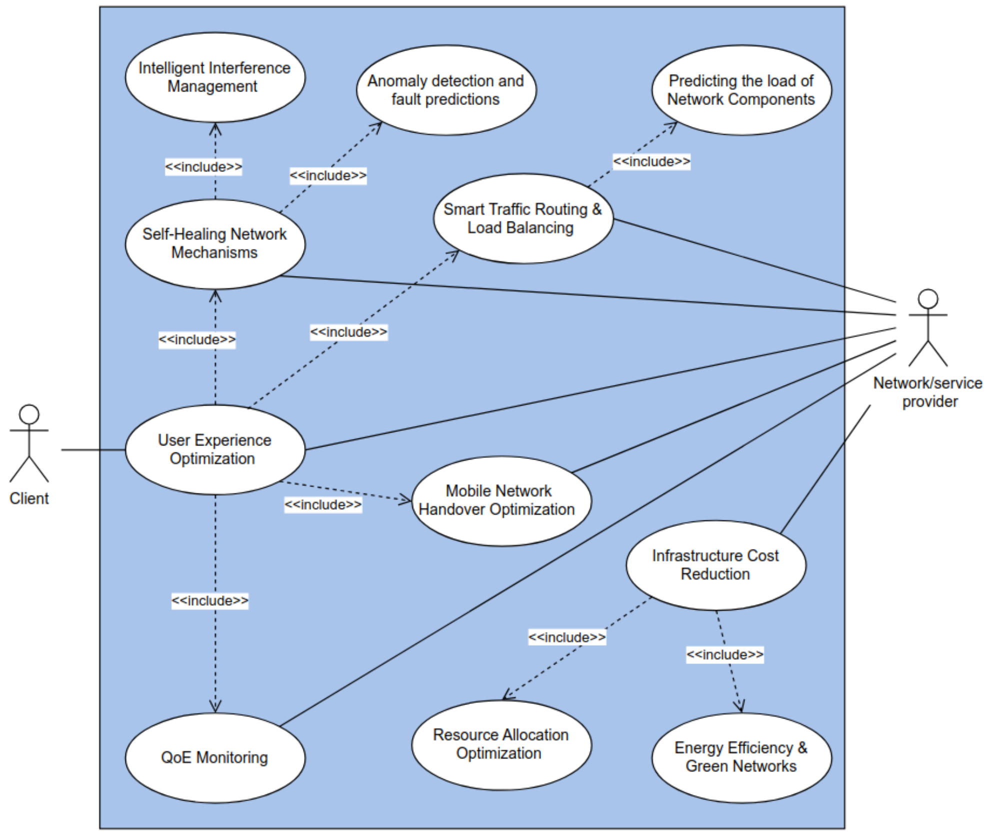
To achieve this, we have identified two key actors:

- **Network/Service Provider**
  - The primary user of the system, responsible for implementing and managing network optimization processes.
- **Service Client**
  - The end-user or customer who benefits indirectly from the improved network performance resulting from the system's optimizations.

In this model, the network/service provider interacts directly with the system, while service clients experience the benefits of enhanced network efficiency and automation.

4.3 Identified Use Cases

The use case diagram presented in Figure 3 shows several possible use cases that could be part of the system, involving both the client and the network/service provider. These include things like smart traffic routing, fault prediction, and energy efficiency. However, due to limited resources and a tight schedule, we decided to focus on implementing just one of these use cases for this project. The other use cases were left out of scope for now but could be considered for future development.



Intelligent Interference Management
Anomaly detection and fault predictions
Predicting the load of Network Components
<<include>>
<<include>>
<<include>>
Smart Traffic Routing & Load Balancing
Self-Healing Network Mechanisms
<<include>>
<<include>>
Network/service provider
Client
User Experience Optimization
Mobile Network Handover Optimization
<<include>>
Infrastructure Cost Reduction
<<include>>
<<include>>
<<include>>
QoE Monitoring
Resource Allocation Optimization
Energy Efficiency & Green Networks

Figure 3: Use Case Diagram

4.4 Use Case

So, we chose to explore the Anomaly Detection and Fault Prediction use case, more specifically, focusing primarily on detecting potential attacks in real time, and then identifying them. The system focuses on recognizing malicious activity as it occurs and classifying the type of attack to support a faster and more accurate response.

In addition to relying on historical trends, the system also processes real-time network data to actively monitor for threats and faults, aiming to improve overall network reliability and security.

4.4.1 Scenario

The following scenario illustrates how the chosen use case could be applied in a real-world environment, specifically within a smart factory setting.

A smart factory relies on thousands of IoT sensors and actuators to control and monitor operations. These include environmental sensors that monitor temperature, humidity, and air quality to ensure product quality, as well as automated machines that perform real-time assembly of products.

Suddenly, the system detects an abrupt increase in data traffic from temperature sensors, which could indicate a malfunction or an ongoing cyberattack. If not addressed, this could damage equipment or pose risks to the safety of nearby workers.

In addition, the system also identifies multiple failed authentication attempts by an unknown device trying to access the robotic assembly line control system. This raises concerns about possible intrusion that could compromise the manufacturing process and disrupt production.

In response, the monitoring system triggers alerts, identifying attacks in real time and classifying them to support appropriate countermeasures, such as isolating affected devices, blocking access attempts, and maintaining stable operations.

This scenario is directly related to the use case of **anomaly detection and fault prediction**, allowing the factory to detect attacks as they occur, respond quickly, and maintain operational safety.

4.4.2 Dataset

The dataset we found to test our use case was the UNSW-NB15 dataset [6], developed by the Australian Centre for Cyber Security (ACCS) at University of New South Wales (UNSW) Canberra. It was specifically created to support research in network intrusion detection, offering a modern and realistic collection of network traffic that includes both normal behavior and a wide range of malicious activity.

The dataset was generated using the IXIA PerfectStorm tool, which emulated realistic network traffic scenarios in a controlled environment. This traffic was captured in raw packet format (pcap), providing a complete view of network activity at the packet level. This is the same format used in our system, which analyzes individual packets to detect and classify attacks.

Alongside the pcap files, ground truth labels are provided, which map each connection to a corresponding attack type or mark it as normal. This allowed us to process the raw traffic directly, extract relevant features, and evaluate the performance of our attack detection and classification system.

The dataset includes a variety of attack categories such as Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. This diversity makes it highly suitable for testing intrusion detection systems in both binary and multiclass classification tasks.

For the data processing phase, we referred to the updated NF3 version of the dataset described in [4], which includes a uniform set of standard flow features enriched with temporal fields. These datasets are derived from well-known NIDS benchmarks and are structured to facilitate consistent training and evaluation of ML models across different attack scenarios.

The selected features cover essential aspects of network flow behavior, including addressing, protocol metadata, packet and byte counts, retransmission statistics, and timing indicators. Temporal precision is supported through millisecond-level timestamps for the start and end of each flow. This comprehensive selection balances efficiency and expressiveness while aligning with the practical needs of intrusion detection.

The full list of features used in this study is provided below:

- `FLOW_START_MILLISECONDS`, `FLOW_END_MILLISECONDS` – Timestamps (in milliseconds) indicating when the flow started and ended.
- `IPV4_SRC_ADDR`, `IPV4_DST_ADDR` – Source and destination IPv4 addresses.
- `L4_SRC_PORT`, `L4_DST_PORT` – Source and destination transport layer (L4) port numbers.
- `PROTOCOL` – Transport protocol (e.g., TCP, UDP).
- `L7_PROTO` – Application-layer protocol identifier (e.g., HTTP, DNS).
- `IN_BYTES`, `OUT_BYTES` – Number of bytes sent in each direction.
- `IN_PKTS`, `OUT_PKTS` – Number of packets sent in each direction.
- `TCP_FLAGS`, `CLIENT_TCP_FLAGS`, `SERVER_TCP_FLAGS` – Aggregated TCP flag values seen in the flow (e.g., SYN, ACK).
- `FLOW_DURATION_MILLISECONDS` – Duration of the flow in milliseconds.
- `DURATION_IN`, `DURATION_OUT` – Duration of client-to-server and server-to-client traffic.
- `MIN_TTL`, `MAX_TTL` – Minimum and maximum Time-To-Live (TTL) values observed.
- `LONGEST_FLOW_PKT`, `SHORTEST_FLOW_PKT` – Byte sizes of the longest and shortest packets in the flow.
- `MIN_IP_PKT_LEN`, `MAX_IP_PKT_LEN` – Minimum and maximum IP packet lengths observed.
- `RETRANSMITTED_IN_BYTES`, `RETRANSMITTED_IN_PKTS`, `RETRANSMITTED_OUT_BYTES`, `RETRANSMITTED_-OUT_PKTS` – Counts of retransmitted bytes and packets in each direction.
- `SRC_TO_DST_AVG_THROUGHPUT`, `DST_TO_SRC_AVG_THROUGHPUT` – Average throughput in bits per second for each direction.
- `NUM_PKTS_UP_TO_128_BYTES`, `NUM_PKTS_128_TO_256_BYTES`, `NUM_PKTS_256_TO_512_BYTES`, `NUM_-PKTS_512_TO_1024_BYTES`, `NUM_PKTS_1024_TO_1514_BYTES` – Distribution of packets based on size ranges.

- TCP_WIN_MAX_IN, TCP_WIN_MAX_OUT – Maximum TCP window size in each direction.
- ICMP_TYPE, ICMP_IPV4_TYPE – ICMP type and subtype for flows using ICMP.
- DNS_QUERY_ID, DNS_QUERY_TYPE, DNS_TTL_ANSWER – DNS transaction ID, query type, and response TTL for DNS flows.
- FTP_COMMAND_RET_CODE – FTP server response code for flows using the FTP protocol.
- Label – Binary classification label (0 = benign, 1 = malicious).
- Attack – Multi-class label indicating specific attack type (e.g., DoS, Backdoor).

To provide context on dataset composition, Table 1 summarizes the number of benign and malicious flows per dataset. Table 2 gives a detailed breakdown of flows by attack class, enabling both binary and multi-class classification tasks.

Dataset	**Malicious Flows**	**Benign Flows**	**Total Flows**
NF3-UNSW-NB15	127,693	2,237,731	2,365,424

Table 1: Summary of Flows in the NF3-UNSW-NB15 Dataset

Attack Type	**Flow Count**
DoS	5,980
Reconnaissance	17,074
Backdoor	1,226
Fuzzers	33,816
Exploits	42,748
Analysis	2,381
Generic	19,651
Shellcode	4,659
Worms	158
Benign	2,237,731

Table 2: Distribution of Attack Types in the NF3-UNSW-NB15 Dataset

4.5 System Requirements

This section defines the specifications for the developed system's requirements, which were gathered through weekly meetings with project supervisors and collaborative brainstorming sessions involving all group members. Since the project followed an agile methodology, the requirements list was continuously refined to accommodate evolving objectives and emerging challenges.

The requirements are categorized into two main groups: general pipeline requirements and use case specific requirements.

4.5.1 Functional Requirements

The system's functional requirements are organized into several categories according to the main aspects of the implementation.

Data Collection

- The system should receive the network data via REST API call.
- The system data receiver should support JSON data format.
- The system should use the message bus to integrate the received network data in the pipeline.

Data Processing

- The system should continuously process the raw network data and extract features.
- The system should clean irrelevant fields from the data.
- The system should use ground truth information to label received data accordingly.
- The system should use the message bus to continuously send the processed data to other pipeline components.

Data Storage

- The system should store ground truth data in a time series database.
- The system should store raw network data in the corresponding time series database.
- The system should store processed network data in the data warehouse.
- The system should store inference data in the corresponding time series database.

ML

- The system should apply multiple ML algorithms on training data.
- The system should apply supervised learning techniques using processed data labels.
- The system should test and evaluate all the ML models after training and select the best one to be deployed.
- The system should have a model deployment mechanism.
- The deployed model should perform inference on real-time processed network data.
- The system should automate continuous training, testing, and deployment as new volumes of training data arrive.
- The system should allow manual training via API call.
- The system should return the respective model training information via callback, after manual training request execution.

Data Visualization

- The system should have a dashboard to visualize pipeline network metrics.
- The dashboard should contain relevant information resulting from network data analytics procedures.
- The dashboard should contain model inference results on live network data.

5G Integration

- The system should collect and expose network data, structuring the payload accordingly with respective 5G NFs on API calls.
- The system should be able to subscribe to event streams from 5GC components via standardized interfaces.
- The system should support the injection of synthetic 5G data for model training purposes.
- The system should support high-precision timestamping to ensure data alignment across 5G components.

Use Case Specific Functional Requirements

- The system should be able to extract features from raw network data.
- The system should have two types of ML algorithms, to train to respectively, identify and classify anomalies.

4.5.2 Non-functional Requirements

The system must meet the following non-functional requirements:

Compliance & Standards

- The system should comply with 3GPP specifications for 5G data collection and exposure.

Scalability

- The system must handle increasing volumes of network data without performance loss.
- The architecture should support the deployment of multiple ML models in parallel.

Performance

- The model inference engine should handle 100 predictions per second with no performance degradation.
- The dashboard should update metrics with a maximum delay of 30 seconds.
- To support real-time analytics, data processing should have minimal latency and a response time below 1 millisecond.

Security

- The system must ensure that all data is kept on-premise.

Maintainability

- The system codebase should follow clean coding principles and modular architecture to ease future updates.
- The system must allow modules to be replaced by others with higher performance, with minimal impact on other modules.
- The system must be easily adaptable for deployment in various network environments.
- The system must follow good MLOps practices, guaranteeing modularity, reproducibility, and full automation of the ML lifecycle.

Reliability & Availability

- If a component fails the rest of the system must remain operational.
- All data should be persisted in respective databases.

Interoperability

- The system must be interoperable, providing APIs and adopting standardized ML frameworks.

Logging and Monitoring

- The system should generate structured logs if errors occur for all pipeline stages.

Use Case Specific Non-Functional Requirements

- The binary model should reach an F1 score percentage above 90%.
- The classification model should reach an F1 score percentage above 90%.

4.6 Architecture

The developed system follows a modular and event-driven architecture designed to support real-time analytics on 5G network data. It is structured to ensure scalability, flexibility, and seamless integration with modern network environments. The architecture was designed to reflect the lifecycle of network data (from initial collection to final insight delivery) while decoupling components to allow independent development, maintenance, and evolution of each module. The pipeline is designed to operate in near real-time, supporting high-throughput scenarios and low-latency requirements typically found in telecommunications environments. It also provides support for both automatic and manual ML workflows, ensuring the solution remains adaptable to evolving analytic needs. This architecture supports the integration of simulated and real data sources, providing a flexible testbed for development and validation. By adopting open technologies and clean interface definitions, it ensures interoperability, maintainability, and compatibility with the standard 5G infrastructure.

Concerning the system's technology model, shown in Figure 4, the following technologies have been identified: Python, Kafka, FastAPI, Scapy, Pandas, nProbe, Scikit-learn, Imbalanced-learn, ClickHouse, InfluxDB, and chronograf.

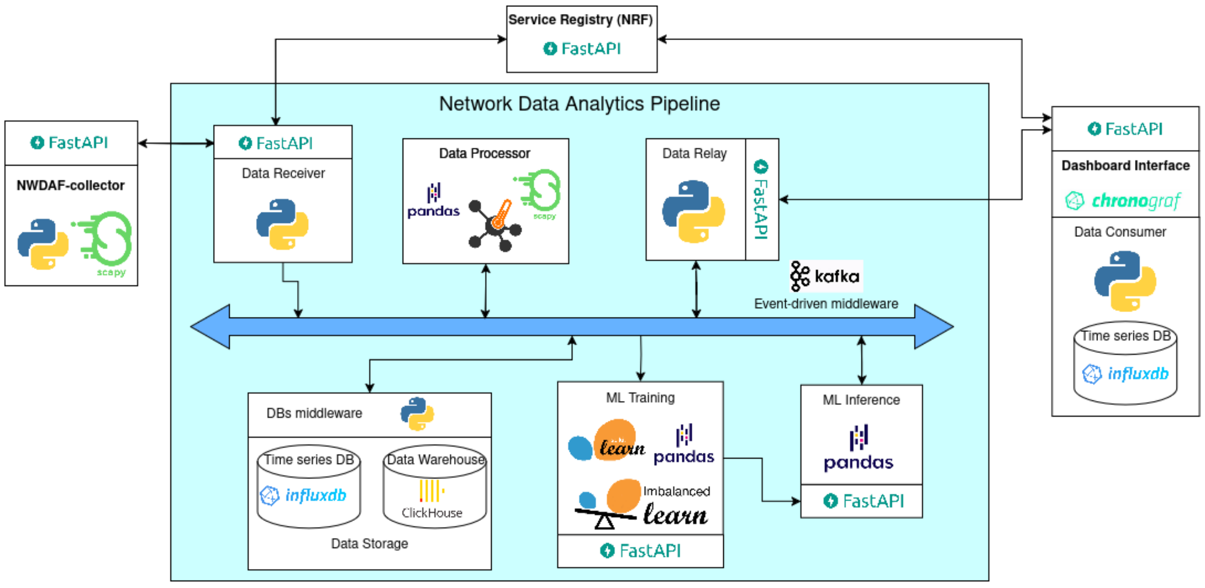


Figure 4: System Architecture

Event-driven middleware: Kafka

The primary way to ensure the pipeline's modules can communicate is by using an event-driven middleware, where the modules can publish data on the appropriate topics. This way, all communication through the middleware is asynchronous. With this approach, every time new data is available to be used by a certain module, it will consume the data in the corresponding topic automatically (the middleware moves the data as close to real-time as possible). Therefore, the middleware acts as an intermediary between different pipeline components, assuring that the modules are decoupled, which is fundamental to ensure reduced interdependence between the system's components and reliable event processing (very important for our pipeline). This separation enhances the system's adaptability, maintainability, and scalability by allowing individual parts to be modified without significantly impacting others.

So, the middleware was implemented using **Kafka**. We chose this technology because it is designed to handle large volumes of data, has low latency, and is highly used in event-driven architectures (architecture shown in the previous figure). Kafka is also excellent for transmitting data in real-time for ML applications such as forecasting and model monitoring. This technology is great for environments where data size growth is high, as it can be scaled horizontally and guarantees message durability (data retention for a defined time). The architecture of this technology also offers great fault tolerance, which is important for our solution, since no message sent between services must be lost.

APIs: FastAPI

In our proposed architecture, both APIs and Kafka play essential and complementary roles in enabling reliable, scalable, and modular communication between the various components of the MLOps pipeline. As the system is designed to operate in real-time and handle high-throughput network data, Kafka serves as a robust, event-driven middleware for decoupling services and efficiently managing streaming data between producers and consumers. Kafka ensures that the pipeline can ingest and process continuous data streams with high availability and fault tolerance.

On the other hand, well-defined APIs are necessary for service orchestration, external system integration, and exposing internal functionality in a controlled and standardized manner. APIs are used in several ways: to enable external systems or network components to push data into the pipeline; to allow various internal modules to invoke each other's functionality; and to make pipeline outputs, such as inference results, accessible to other NFs. Following RESTful principles and contract-driven design practices ensures long-term maintainability and interoperability of these interfaces.

FastAPI is a modern web framework that is fast and used to build APIs in Python and which automatically produces documentation of the implemented APIs (OpenAPI and Swagger). This framework is simple and easy to use, reducing development and maintenance time. It is also important to note that it is compatible with Pytorch and Scikit-learn, facilitating integration into MLOps pipelines. FastAPI is therefore an efficient and fast way to produce APIs, without wasting time documenting them.

NWDAF Collector: Scapy

In the early stages of pipeline development, it is essential to have a controlled and reproducible environment for testing the system's ability to ingest, process, and analyze real-world traffic. To meet this requirement, we developed a custom NWDAF collector that simulates the behavior of a 5GC network function. This module serves as the primary entry point for data into our pipeline, playing a vital role in feeding raw network traffic into the system for further processing and analysis.

The NWDAF collector is responsible for supplying the pipeline with realistic network packets by reading from a dataset composed of multiple .pcap files. These packet capture files contain actual recorded traffic captured in various network scenarios. To extract and manipulate this packet data, the NWDAF collector leverages **Scapy**, a powerful Python library used for crafting, parsing, and analyzing network packets. Scapy excels at dissecting packet headers, reconstructing protocol stacks, and extracting metadata required by downstream components. In this context, Scapy is used to interpret the contents of the .pcap files. The flexibility of Scapy allows the collector to simulate complex traffic patterns with possible attack information.

Once the packets are parsed and prepared, the NWDAF collector exposes this raw traffic data through a FastAPI interface, ensuring seamless integration with the pipeline's data ingestion component (Data Receiver).

Data Receiver and Data Relay: Python

To ensure seamless communication between the pipeline and the 5GC, two critical components were developed: the **Data Receiver** and the **Data Relay**. These modules serve as the primary input and output interfaces of the system, enabling bidirectional integration with NFs and external services.

The Data Receiver is responsible for ingesting data from the 5GC. It acts as a listening service, exposed via a FastAPI endpoint, that receives incoming data from various 5G NFs. These payloads, often structured in JSON format and aligned with 3GPP-compliant data models, are immediately pushed to Kafka topics for downstream consumption. This design ensures that as soon as a core function exposes network data, it is received by the pipeline in near real time and made available to the processing and ML modules. Python was chosen to implement this service due to its simplicity and strong ecosystem support for working with network data and asynchronous APIs. On the other side of the pipeline, the Data Relay is responsible for exposing processed data and inference results back to the 5GC or any other consumer system. This component formats and serves the analytics output through another FastAPI interface. By exposing these results via RESTful endpoints, the Data Relay enables external NFs, dashboards, or orchestration tools to access live insights generated by the ML models and the data processor. The use of Kafka ensures that all outgoing data is decoupled from upstream processing, maintaining system flexibility and scalability.

Together, the Data Receiver and Data Relay form the external communication boundary of the pipeline. They ensure that the system remains interoperable, standards-compliant, and capable of operating in real 5G network environments. Both are lightweight Python services, easy to deploy and maintain, and tightly integrated into the event-driven flow of the pipeline.

Data Processor: Pandas, nProbe, and Scapy

The data processor is a key component of our pipeline, responsible for transforming raw traffic data collected from the network into clean, structured, and enriched information that can be used by downstream modules and external systems. Most network data is not directly usable in its raw form, especially for tasks like training ML models or feeding analytics dashboards. This module ensures that the incoming traffic (often noisy, unstructured, and high-volume) is parsed, filtered, and converted into valuable insights. To accomplish this, our data processor uses a combination of **Scapy**, **nProbe**, and **Pandas**.

We integrate nProbe, a high-performance flow exporter and collector that converts packet-level data into rich flow-level metadata. nProbe is used to extract **NetFlow features**, which are essential for understanding traffic behavior at an aggregated level. These features include flow duration, number of packets and bytes exchanged, TCP flag summaries, average packet size, flow directionality, protocol distribution, and inter-arrival times. By exporting this information in NetFlow or IPFIX formats, nProbe adds semantic meaning and behavioral context to the data, which is essential for tasks like anomaly detection, traffic classification, and usage profiling. Additionally, nProbe supports advanced traffic analysis capabilities such as application protocol recognition through deep packet inspection (DPI), TLS fingerprinting, and classification using NBAR2, all of which enrich the dataset used downstream.

Once data is parsed and enriched, it is structured and prepared for analytics using Pandas, a widely adopted data analysis library in Python. Pandas provides a robust set of tools for cleaning, transforming, and aggregating structured data. In our pipeline, Pandas handles missing values, normalizes data formats, creates aggregated views (e.g., per-user, per-flow, or per-network slice), and encodes features suitable for ML models. With its powerful DataFrame abstraction and interoperability with the rest of the data science ecosystem, Pandas enables efficient data wrangling and fast prototyping of ML-ready datasets.

Data storage: influxDB and ClickHouse

It is a requirement of the project that our pipeline stores the data instead of using an external service. So, to accommodate this necessity, we have a data storage module where all data is kept on-premise. This module has a middleware and two databases. The middleware agent is responsible for fetching the data present in the event-driven middleware (Kafka), and deciding in which database it should store them. This decision is made according to the nature of the data. The raw data fetched from the 5GC NFs is stored in our time series database (influxDB) and the processed data is stored in the data warehouse (ClickHouse).

- **InfluxDB** is a high-performance, open-source time series database. It is optimized to handle high write rates and real-time analysis, making it ideal for use cases such as IoT sensor data and monitoring. It also has a query language similar to SQL but specified for time series data. In this way, influxDB is an excellent choice to integrate into our pipeline and use as the main source of storage for data received from the network, since this data comes from IoT sensors and there is great importance in maintaining the chronological order of the time record in which it was collected.
- **ClickHouse** is an open-source column-oriented database that stands out for its high performance as a data warehouse for storing analytical and time series data. It has a query language similar to SQL and is optimized for high-speed queries on large data sets, making it an excellent choice for analyzing large volumes of data. It is also important to mention its efficient storage with advanced compression techniques, which reduce the cost of storage (something crucial when working with large volumes of data). Therefore, ClickHouse is the perfect choice for storing processed data, ready to be used to train the ML model, and data from the model's inference, which is then distributed to data consumers.

ML training and inference: Scikit-learn and Imbalanced-learn

To ensure accurate and reliable model predictions, our pipeline includes a dedicated ML training and inference module. This component is responsible for training and testing ML models using the processed data stored in the data warehouse (ClickHouse) before the models are deployed for real-time inference.

For the initial version of the pipeline, we chose to use **Scikit-learn** and **Imbalanced-learn**, both of which are robust and mature Python libraries suitable for building classical ML solutions. These technologies offer a solid foundation for supervised and unsupervised learning tasks such as classification, regression, and clustering, all of which are critical in network data analytics.

Scikit-learn is an open-source ML library built on top of NumPy, SciPy, and Matplotlib. It includes a wide range of efficient tools for ML and statistical modeling, including support vector machines, random forests, gradient boosting, k-means, and more. Its ease of use, extensive documentation, and integration with other Python libraries make it an ideal choice for prototyping and deploying models in production.

Imbalanced-learn is a complementary library that provides techniques specifically designed for imbalanced datasets, which are common in real-world network traffic (e.g., rare failure events or anomalies). It supports resampling methods such as SMOTE (Synthetic Minority Over-sampling Technique), undersampling, and combined strategies that help improve model performance on skewed datasets.

By using these two libraries together, we ensure the development of balanced, high-performing models that are well suited for network traffic prediction and anomaly detection. These models are then deployed via FastAPI, making their predictions available in real-time to other components in the pipeline or external consumers.

Dashboard interface: Chronograf

Before trying to integrate the pipeline in the 5GC network, we need to simulate the behavior of a network function that would consume the data produced by our pipeline (process and inference data) so we can validate it. The easiest way to visualize this data is to use a dashboard where we can display all the metrics. Instead of wasting time building a dashboard from scratch, we can use technologies that make the process easier and faster. So, on that note, we decided to use Chronograf developed by InfluxData.

Chronograf is a data visualization and monitoring tool used to analyze various metrics and records obtained in real-time. It offers a complete dashboarding solution for visualizing data. With it, we can easily clone a pre-canned dashboard (over 20 pre-canned dashboards). This saves a lot of development time since you don't have to make a complete interface from scratch with a specialized framework. Chronograf allows us to use Flux or the InfluxQL queries to fetch and analyze data. This tool makes it easy to develop dynamic and customizable interfaces without a great deal of effort and time, making it the perfect choice for simulating the operation of a 5GC network function that consumes the data provided by the pipeline.

4.7 Docker Deployment: Containerization and Orchestration of the Pipeline

To ensure modularity, portability, and ease of deployment, the entire pipeline architecture was containerized using Docker. Each module of the system was isolated into its own container or grouped with related components to maintain logical cohesion. This separation allows for independent management, scaling, and troubleshooting of each part of the system while also facilitating replication across different environments. The current deployment setup includes the following main containers:

- **Data Collector:** This container simulates a network function from the 5GC using Scapy. It is responsible for feeding raw packet data into the pipeline.
- **Kafka Middleware:** A set of containers responsible for message brokering, enabling asynchronous communication between all pipeline components. These include Kafka, Zookeeper, and supporting services for managing topics and partitions.
- **Data Processor:** This container runs the processing logic, which includes parsing, cleaning, transforming, and aggregating raw data.
- **Databases:** Three separate containers are used for on-premise storage, each serving a distinct role in the data lifecycle:
  - **InfluxDB (Raw Data):** Stores the raw, time series data collected from the network before processing. This instance is optimized for high write throughput and chronological consistency, crucial for time-sensitive network telemetry.
  - **ClickHouse:** Holds the processed and aggregated data that is used for analytics, ML training, and inference. Its columnar storage and query efficiency make it ideal for handling large volumes of structured, high-dimensional data.

  - **InfluxDB (Inference Results):** A dedicated instance used to store the output of the ML inference process. This separation allows for seamless integration with Chronograf, enabling real-time visualization of prediction results.

- **ML Training:** This container is responsible for training and validating ML models. The container handles all preprocessing steps required for model training and supports experimentation with various classical ML algorithms.

- **ML Inference:** This container performs real-time or batch inference using the trained models. This separation ensures that inference workloads are decoupled from training, improving reliability and scalability during deployment.

- **API Services:** Exposes APIs built with FastAPI that handle both ingestion from external sources and delivery of processed results or inference outputs. It ensures secure, standardized interaction with external consumers or other NFs.

- **Dashboard Interface:** The dashboard is deployed in its own container and runs Chronograf, which provides real-time visualization of metrics and analytics generated by the pipeline.

Docker Compose: Coordinated Multi-Container Management

To orchestrate all these services and ensure their correct initialization and communication, Docker Compose was employed. This tool facilitates the coordinated startup of all containers, manages inter-container networking, and enforces dependency order.

Docker Compose ensures that environment variables are consistently defined and managed across services, making the setup highly reproducible. It also enables periodic health checks, automatic container restarts upon failure, and centralized management of persistent volumes. In production-like environments, this orchestration streamlines updates, provides robust container-level logging, and simplifies the deployment and scaling of the entire pipeline stack.

In summary, using Docker and Docker Compose allowed for a clean, modular deployment strategy that simplifies the system's maintenance, enhances its portability, and supports agile development and testing cycles across different environments.

4.8 Data Flow

Knowing how data flows in the pipeline is extremely important for properly understanding the implementation. Each component of the system interacts with other components by messaging, using an event-driven middleware implemented with kafka, and by API calls, using Fast Api endpoints (shown in Figure 7).

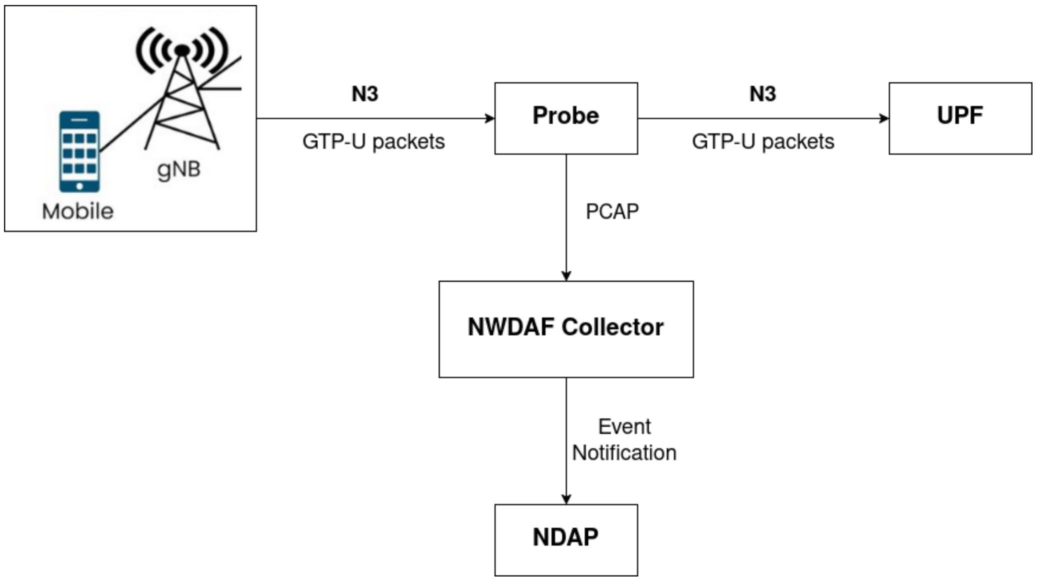
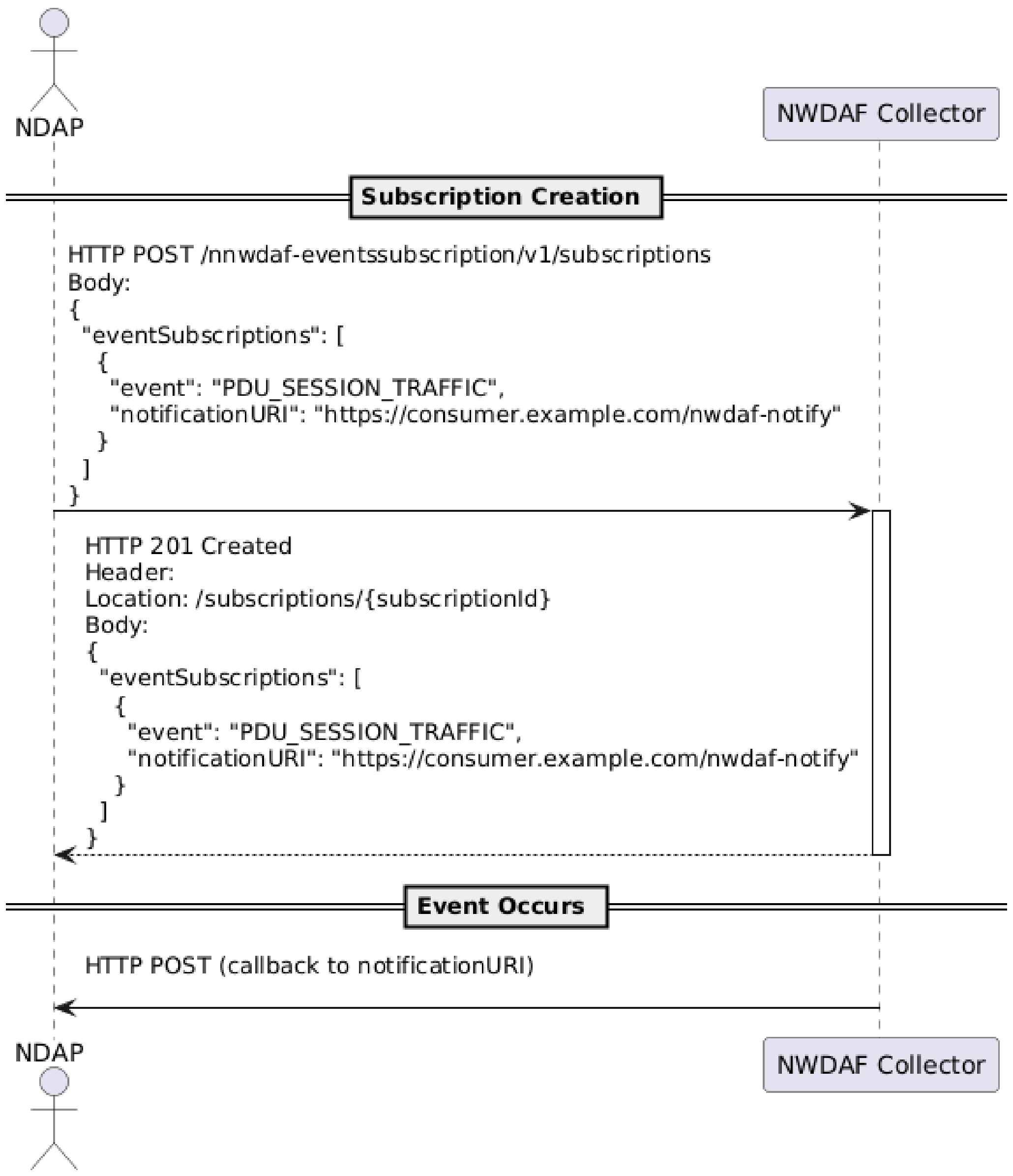


Figure 5: Simulation of traffic data extraction in a 5G core network

The system has a Kafka message broker with 4 topics: RAW_NETWORK_DATA_RECEIVED, RAW_-NETWORK_DATA, PROCESSED_NETWORK_DATA, INFERENCE_DATA. These topics are useful, for the corresponding components consume from or produce to, data. There two types of APIs used, one inside the pipeline to deploy the best model for inference, and the others correspond to 5G NFs requests.

To simulate the network data that is supposed to be received from the 5G network core, the dataset mentioned before was used. This dataset had pcap files, with millions of captured packets. Our data producer (NWDAF-collector) transforms this data into a format accordingly to a 5G network function information.

The NWDAF Collector plays a crucial role in the simulation of network data acquisition. As shown in Figure 5, it emulates the behavior of a real NWDAF-like function by capturing packets between the gNB and the UPF on the N3 interface. These packets are encapsulated in GTP-U format, and a probe component is responsible for intercepting and duplicating this traffic, saving it in the form of PCAP files. The NWDAF Collector exposes this data through standard endpoints. This process effectively simulates the real-time flow of data from the 5GC network into an analytics system.



```
HTTP POST /nnwdaf-eventssubscription/v1/subscriptions
Body:
{
  "eventSubscriptions": [
    {
      "event": "PDU_SESSION_TRAFFIC",
      "notificationURI": "https://consumer.example.com/nwdaf-notify"
    }
  ]
}
```

```
HTTP 201 Created
Header:
Location: /subscriptions/{subscriptionId}
Body:
{
  "eventSubscriptions": [
    {
      "event": "PDU_SESSION_TRAFFIC",
      "notificationURI": "https://consumer.example.com/nwdaf-notify"
    }
  ]
}
```

Figure 6: NWDAF Event Subscription and Notification Flow

To enable interoperability and compliance with 5G SBA principles, the pipeline components must dynamically discover and interact with other NFs. This begins with the pipeline registering itself with the NRF. The registration includes specifying its service capabilities and endpoints. Once registered, the pipeline can issue NF discovery requests to the NRF to locate other NFs (like the NWDAF Collector) that expose needed services. This dynamic discovery mechanism ensures that the pipeline remains loosely coupled and scalable, and can adapt to varying network environments.

Following successful discovery, the next step involves subscribing to events exposed by the NWDAF Collector. This is illustrated in the second diagram. A consumer component, such as NDAP or any analytic microservice, initiates a subscription creation request via HTTP POST to the /nnwdaf-eventssubscription/v1/subscriptions endpoint. The request specifies the type of event it is interested in, such as PDU_SESSION_TRAFFIC, and provides a notificationURI to which the Collector will send data. Upon successful subscription, the NWDAF Collector responds with a 201 Created status and provides the subscription details. Later, when the event condition is met (a batch of traffic data becomes available), the Collector triggers an HTTP POST callback to the consumer's notificationURI, pushing the batch data (shown in Figure 6).

This architecture simulates a standard-compliant 5G analytics interaction loop, enabling seamless data ingestion, transformation, and downstream processing.

The Data Receiver component ingests network traffic data that has already been preformatted appropriately. This data is then forwarded directly to the Kafka topic named RAW_NETWORK_DATA, acting as the primary entry point into the processing pipeline. Two downstream components subscribe to this topic: the Data Storage Middleware and the Data Processor. Each of these serves a distinct function, with the former responsible for archiving the raw data and the latter for real-time analytical transformation.

The Data Storage Middleware, in its first role, transforms the raw network data into a format suitable for time-series storage by aligning it with InfluxDB's structural requirements. Specifically, it uses the packet timestamp as the index, thereby enabling efficient chronological querying and long-term retention. This archival of raw network traffic is critical, as it provides a fallback source for data recovery in case of processing failures and supports future exploratory analysis or forensic investigations.

Simultaneously, the Data Processor operates as the core of the analytical pipeline. It consumes the same raw network traffic and aggregates the packets into dynamic batches. When a predefined threshold is reached, a batch is encapsulated into a temporary PCAP file. This file is then processed using nProbe, a tool that converts packet-level data into flow-level records. nProbe extracts 43 distinct flow features from each batch, enriching the data further by attaching ground truth labels where applicable. The resulting structured dataset is then forwarded to the PROCESSED_NETWORK_DATA Kafka topic, making it available for further consumption and downstream tasks.

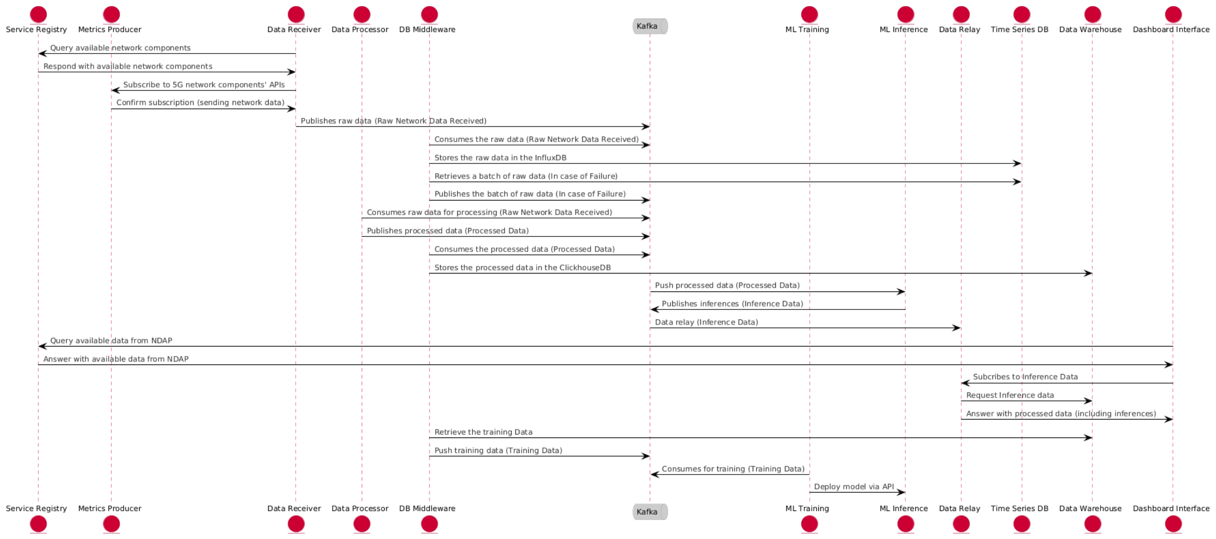

Service Registry
Metrics Producer
Data Receiver
Data Processor
DB Middleware
Kafka
ML Training
ML Inference
Data Relay
Time Series DB
Data Warehouse
Dashboard Interface
Query available network components
Respond with available network components
Subscribe to 5G network components' APIs
Confirm subscription (sending network data)
Publishes raw data (Raw Network Data Received)
Consumes the raw data (Raw Network Data Received)
Stores the raw data in the InfluxDB
Retrieves a batch of raw data (In case of Failure)
Publishes the batch of raw data (In case of Failure)
Consumes raw data for processing (Raw Network Data Received)
Publishes processed data (Processed Data)
Consumes the processed data (Processed Data)
Stores the processed data in the ClickhouseDB
Push processed data (Processed Data)
Publishes inferences (Inference Data)
Data relay (Inference Data)
Query available data from NDAP
Answer with available data from NDAP
Subcribes to Inference Data
Request Inference data
Answer with processed data (including inferences)
Retrieve the training Data
Push training data (Training Data)
Consumes for training (Training Data)
Deploy model via API

Figure 7: System Data Flow

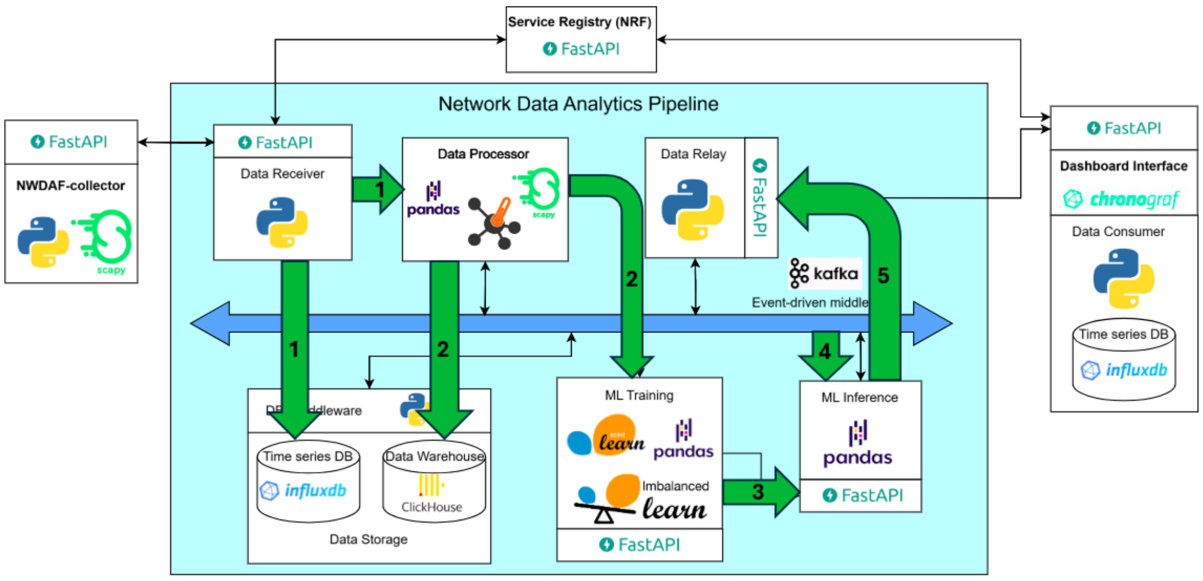


Figure 8: System Data Flow (Architectural View)

From here, the Data Storage Middleware takes on a second responsibility. It now consumes the enriched flow data and converts it into a format compatible with the ClickHouse database, the system's primary data warehouse. ClickHouse, known for its high-performance analytical capabilities, stores this data persistently to ensure rapid querying and reliable backup for processed traffic information. This stored dataset forms the basis for ML model training and evaluation.

The ML Training component is tasked with building predictive models for attack detection using the processed network data. At startup, it loads historical data from the ClickHouse warehouse into memory using the Pandas library. This component then conducts a preprocessing phase where specific features are removed to minimize noise and potential bias. For binary classification tasks, the features excluded from training include: FLOW_START_MILLISECONDS, FLOW_END_MILLISECONDS, IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, ICMP_TYPE, ICMP_IPV4_TYPE, DNS_QUERY_ID, DNS_QUERY_TYPE, DNS_TTL_ANSWER, FTP_COMMAND_RET_CODE, Attack, and id. For multiclass classification, the same features are removed except for the Attack label, which is retained as the target variable.

The model training process employs several supervised learning algorithms. For binary classification, the models used include Random Forest, Gradient Boosting, Multilayer Perceptron (MLP) neural networks from the Scikit-learn library, and the XGBoost classifier from the xgboost library. The same set of models, excluding XGBoost, are used for multiclass classification, utilizing their respective multiclass implementations. To address the severe class imbalance, where benign traffic heavily outweighs attack instances, the system applies oversampling and undersampling techniques such as SMOTE and random undersampling. This helps reduce overfitting and improves generalization performance.

The training dataset is split into 80% for training and 20% for testing, using stratified sampling to maintain class distribution across subsets. Each algorithm is trained sequentially using the same preprocessed data. After training, each model is evaluated using its classification report, and key metrics are computed. The primary metric used for model selection is the F1 score, as it balances precision and recall and is particularly well suited for imbalanced classification problems like intrusion detection. For both binary and multiclass tasks, the model achieving the highest F1-score is selected for deployment.

Once the best models are identified, they are serialized using Python's pickle module. This serialization format allows efficient transfer of model objects to the ML Inference component (shown in Figure 8). Deployment is performed dynamically through HTTP POST requests to API endpoints exposed by the ML Inference service, replacing the currently deployed models with the newly trained ones in real time.

The ML Inference component continuously consumes data from the PROCESSED_NETWORK_DATA Kafka topic and performs real-time attack detection. It applies the deployed models to classify each network flow as either 'benign' or 'attack' for binary detection and, in the case of detected attacks, assigns one of several predefined categories including 'Fuzzers', 'Analysis', 'Backdoors', 'DoS', 'Exploits', 'Generic', 'Reconnaissance', 'Shellcode', or 'Worms'. Once a prediction is made, the corresponding labels are appended to each data row. The labeled data is then published to the INFERENCE_DATA Kafka topic, where it becomes available for further analysis or integration with downstream systems such as dashboards, alert engines, or response mechanisms. This cycle repeats continuously, enabling the system to maintain up-to-date threat intelligence and detection capabilities in a live network environment.

Once the inference data is generated by the ML Inference module, it is routed to the Data Relay component, which is tasked with exposing the analytics results to interested external entities. This exposure process emulates the 3GPP-compliant NWDAF behavior by enabling third-party NFs to subscribe to specific analytics events. To simulate such behavior, a Data Consumer (representing an NF consumer) first registers itself with the NRF via FastAPI endpoints. This step ensures service discovery and aligns with the SBA defined in the 5GC.

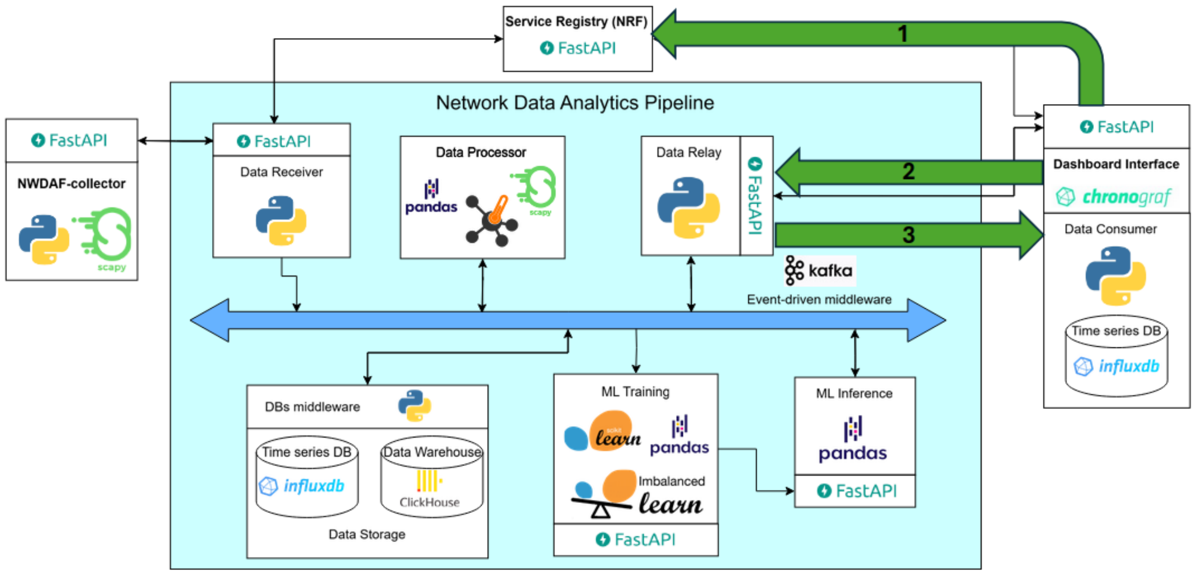
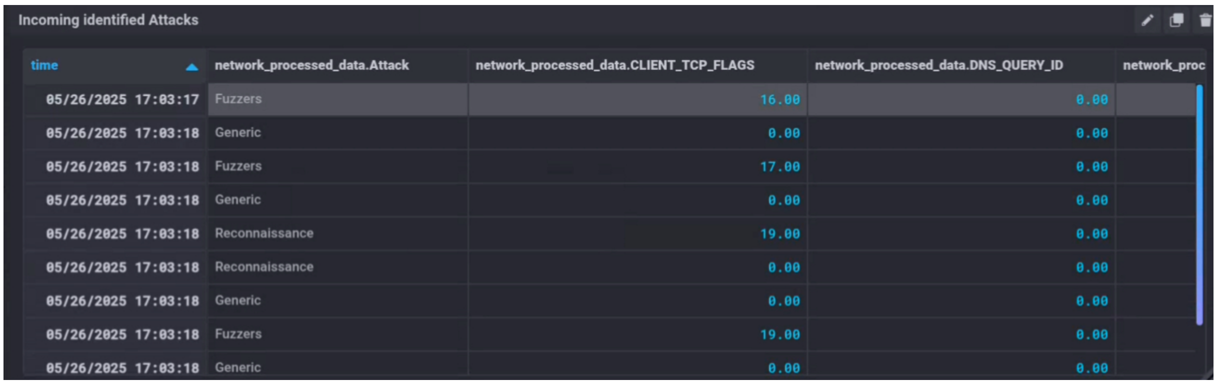


Figure 9: Service Exposure Flow

After registration, the Data Consumer proceeds to subscribe to the pipeline's analytics feed using the endpoint /nnwdaf-eventssubscription/v1/subscriptions. The subscription includes a JSON body indicating interest in the PDU_SESSION_TRAFFIC event type and providing a notificationURI to which the pipeline should send notifications. This interaction simulates the standardized procedure used by real-world NFs to request data from NWDAF (shown in Figure 9).

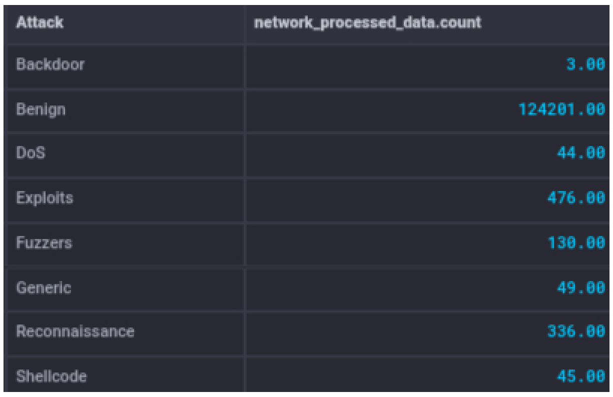
Once the subscription is accepted and acknowledged, the Data Relay begins delivering the inference results in batch mode to the consumer's specified callback URI via HTTP POST requests. These payloads contain the processed insights derived from observed traffic patterns, effectively mimicking NWDAF's real-time and historical data exposure. The combination of NRF registration, subscription management, and asynchronous event notification demonstrates how the pipeline supports real-time analytics exposure, a key enabler for intelligent network automation in 5G.

In the Dashboard Interface component, data is stored in a dedicated time-series database using InfluxDB, chosen for its native integration with Chronograf, the selected data visualization and analysis tool for this implementation. Chronograf enables seamless interaction with time-series data, allowing it to be easily displayed in both tabular and graphical formats. This greatly simplifies the process of monitoring and analyzing network traffic. Among the key metrics visualized are the most recent network flows filtered specifically for those labeled as attacks (shown in Figure 10), the distribution and frequency of each attack type (shown in Figure 11), and various other insights essential for real-time threat monitoring and historical trend analysis, as illustrated in Figure 12.


Incoming identified Attacks

time	network_processed_data.Attack	network_processed_data.CLIENT_TCP_FLAGS	network_processed_data.DNS_QUERY_ID	network_proc
05/26/2025 17:03:17	Fuzzers	16.00	0.00	
05/26/2025 17:03:18	Generic	0.00	0.00	
05/26/2025 17:03:18	Fuzzers	17.00	0.00	
05/26/2025 17:03:18	Generic	0.00	0.00	
05/26/2025 17:03:18	Reconnaissance	19.00	0.00	
05/26/2025 17:03:18	Reconnaissance	0.00	0.00	
05/26/2025 17:03:18	Generic	0.00	0.00	
05/26/2025 17:03:18	Fuzzers	19.00	0.00	
05/26/2025 17:03:18	Generic	0.00	0.00	

Figure 10: Last received attacks



Attack	network_processed_data.count
Backdoor	3.00
Benign	124201.00
DoS	44.00
Exploits	476.00
Fuzzers	130.00
Generic	49.00
Reconnaissance	336.00
Shellcode	45.00

Figure 11: Attack frequency by type

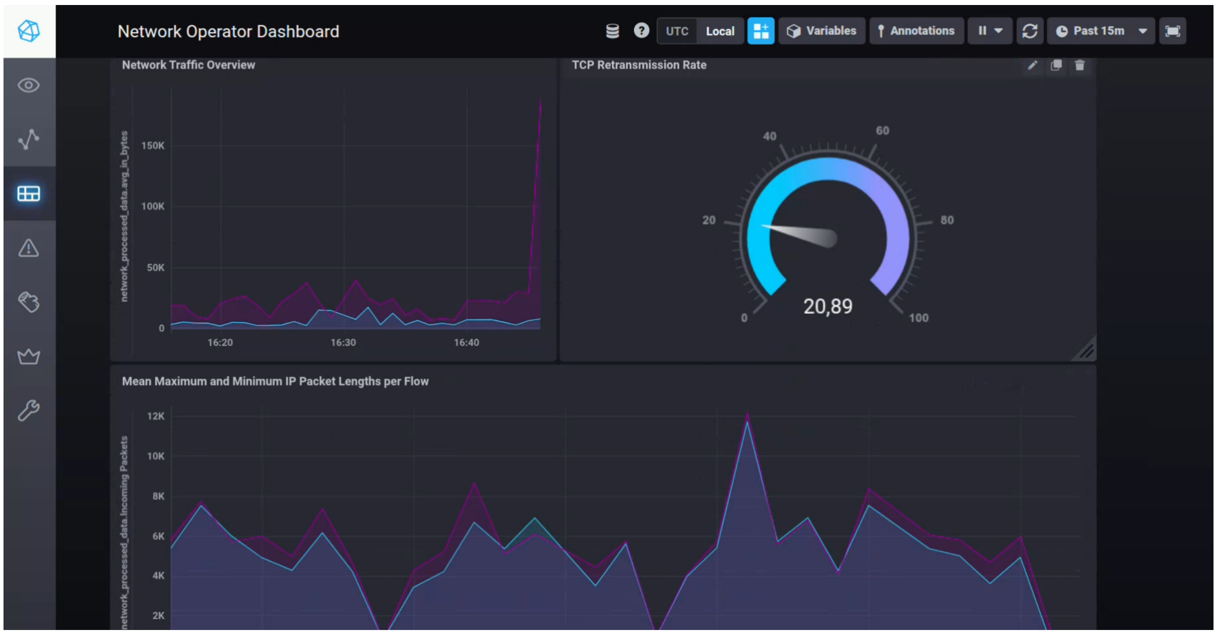
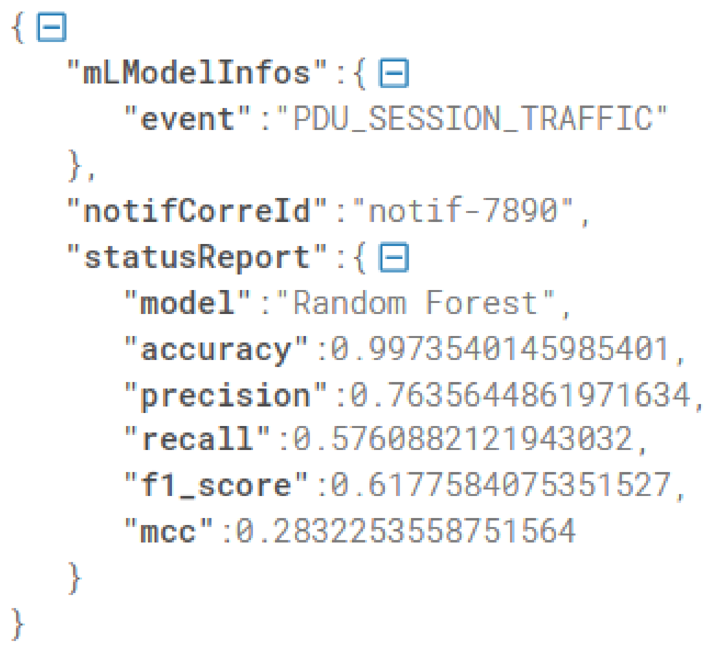

Network Operator Dashboard
UTC
Local
Variables
Annotations
Past 15m
Network Traffic Overview
network_processed_data.avg_in_bytes
150K
100K
50K
0
16:20
16:30
16:40
TCP Retransmission Rate
0
20
40
60
80
100
20,89
Mean Maximum and Minimum IP Packet Lengths per Flow
network_processed_data.Incoming Packets
12K
10K
8K
6K
4K
2K

Figure 12: Dashboard with data analytics metrics (Network Traffic Overview, TCP Retransmission Rate)

The implementation also includes an additional data flow that supports manual model training triggered by an API call. The ML Training component exposes a POST API endpoint at /nnwdafmlmodeltraining/v1/subscriptions, which allows external services to request the training of a specific ML model. This endpoint requires the model ID as a parameter in the HTTP POST request, and it initiates the training process for the corresponding model, applicable only to binary classification models. Upon completion of the training, the component returns a set of performance metrics via a callback mechanism. These metrics include the model name, accuracy, precision, recall, F1-score, and the Matthews Correlation Coefficient (MCC), providing a comprehensive evaluation of the model's performance, as shown in Figure 13.



```
{
   "mLModelInfos":{
      "event":"PDU_SESSION_TRAFFIC"
   },
   "notifCorreId":"notif-7890",
   "statusReport":{
      "model":"Random Forest",
      "accuracy":0.9973540145985401,
      "precision":0.7635644861971634,
      "recall":0.5760882121943032,
      "f1_score":0.6177584075351527,
      "mcc":0.2832253558751564
   }
}
```

Figure 13: Training API response schema

5 Results

The developed system not only fulfills the functional goals defined for the selected use case but also successfully meets a wide range of non-functional requirements that are essential for real-world deployment in dynamic 5G environments. Starting with compliance and standards, the architecture was designed to emulate an NWDAF-like component, respecting 3GPP specifications and service-based architecture principles. RESTful APIs were implemented using FastAPI, including mock interactions with NRF and NWDAF event subscription endpoints, ensuring compatibility with modern 5GC systems.

Scalability in the system was supported through the integration of Kafka as the event-driven middleware. Kafka enables asynchronous communication between decoupled components and supports horizontal scaling, making the architecture suitable for handling large volumes of data and deploying multiple ML models in parallel. While formal performance benchmarking was not conducted, communication between components was thoroughly verified, and message transmission times were observed to remain consistently low. We manually varied the volume of consumed data and observed that the system maintained stable performance across different loads. During system integration, component communication was observed to be efficient, with low-latency data flow across the pipeline. The dashboard consistently displayed updates in near real-time, with refresh times staying comfortably within the expected 30 second responsiveness window.

In terms of security, all collected, processed, and stored data remained on-premise, ensuring data sovereignty and compliance with privacy and protection requirements.

From a maintainability perspective, the system was designed with a modular architecture that supports independent development, replacement, and scaling of each component. Modules are separated and loosely coupled via Kafka and APIs, facilitating future updates and improvements.

Reliability and availability were also core concerns. Kafka's message durability guarantees that no data is lost during service interruptions, and Docker's health check and restart policies ensure that isolated component failures do not disrupt the entire system. All critical data was persisted using structured storage solutions, and structured logging was implemented to support observability and debugging.

To further support operational transparency, logging and monitoring capabilities were integrated across all core modules. These logs capture processing activity, model behavior, and system health, laying the groundwork for future integration with standardized monitoring solutions.

Finally, the effectiveness of the system was validated using a selected anomaly detection use case. While the machine learning performance metrics fell short of expectations, achieving an F1-score below the targeted 90% we believe there is strong potential for improvement. As more data becomes available, particularly additional examples of anomalies, the model's performance is expected to improve significantly. This is due to the increased training data helping to address the current class imbalance, which is a known limitation in early-stage anomaly detection systems.

Under these experimental conditions, two distinct executions were performed to evaluate the evolution of model performance in detecting and classifying network attacks. The first execution, initiated at the beginning of the dataset and running for a longer duration, enabled the extraction of a greater number of flows, providing a more comprehensive view of how algorithm performance evolved over time. The second execution, however, commenced at the 3.pcap file, which contains a significantly higher concentration of anomalous data than prior segments. This alternative approach aimed to assess the impact of early exposure to anomaly-rich data on machine learning models.

For each execution, two tables and two plots were produced to facilitate a detailed analysis. The first table presents the number of detected attacks relative to the total number of flows (Table 3 and 5), allowing for an evaluation of attack ratios across different intervals. By examining the progression of attack detection over time, insights can be drawn regarding the adaptability and sensitivity of the algorithms as they encounter more data.

The second table provides a more granular breakdown by categorizing attacks according to their type across flow intervals (Table 4 and 6). This data serves as a foundation for assessing the effectiveness of multiclass classification models, which aim to correctly identify specific attack types. The temporal evolution of classification accuracy offers valuable perspectives on the models' capability to refine their understanding of distinct threat patterns.

Complementing these tabular representations, two plots illustrate the variation in F1-score across training and testing iterations. The first, shown in Figure 14 and 16, plot focuses on binary attack detection, evaluating the performance of Random Forest, Gradient Boosting, MLP Neural Networks, and XGBoost in distinguishing between normal and malicious traffic. The second plot (Figure 15 and 17) extends the scope to multiclass attack classification, measuring the precision of each algorithm in categorizing attacks at intervals of 5000 flows.

Through a comparative evaluation of both executions, important conclusions can be drawn regarding the effects of dataset composition and retraining frequency on predictive accuracy, and also take conclusions of each algorith suitability for the task.

5.1 Execution 1

Number of Flows	**Number of Attacks**
5000	10
10000	12
15000	52
20000	96
25000	149
30000	187
35000	209
40000	212
45000	212
50000	212
55000	212
60000	212
65000	212
70000	212
75000	212
80000	212
85000	212
90000	212
95000	212
100000	228
105000	302
110000	345
115000	390
120000	423
125000	473
130000	501
135000	550
140000	589
145000	628
150000	659
155000	736
160000	758
165000	781
170000	809
175000	854
180000	900
185000	961
190000	1005
195000	1208
200000	1291
205000	1373
210000	1445

Table 3: Number of Attacks Detected per Flows Number - Execution 1

	Fuzzers	**Reconnaissance**	**Shellcode**	**Backdoor**	**DoS**	**Exploits**	**Generic**	**Worms**
5000	2	1	0	0	2	5	0	0
10000	2	2	0	0	2	6	0	0
15000	2	11	4	0	4	27	3	0
20000	10	20	6	0	7	47	5	0
25000	24	32	9	0	12	66	5	0
30000	48	40	9	0	13	69	7	0
35000	60	45	10	0	14	71	8	0
40000	60	50	10	0	17	76	14	0
45000	60	50	10	0	17	76	14	0
50000	60	50	10	0	17	76	14	0
55000	60	50	10	0	17	76	14	0
60000	60	50	10	0	17	76	14	0
65000	60	50	10	0	17	76	14	0
70000	60	50	10	0	17	76	14	0
75000	60	50	10	0	17	76	14	0
80000	60	50	10	0	17	76	14	0
85000	60	50	10	0	17	76	14	0
90000	60	50	10	0	17	76	14	0
95000	60	50	10	0	17	76	14	0
100000	60	50	10	0	17	76	14	0
105000	85	58	12	0	21	103	22	0
110000	92	66	13	0	25	125	23	0
115000	92	75	14	0	38	146	24	0
120000	92	84	14	0	39	167	26	0
125000	100	101	20	1	39	183	28	1
130000	108	107	20	1	40	195	29	1
135000	110	121	22	1	45	216	34	1
140000	110	130	22	1	48	237	40	1
145000	125	137	22	1	49	253	40	1
150000	125	138	23	1	54	274	43	1
155000	131	149	23	1	59	324	48	1
160000	137	153	24	1	62	332	48	1
165000	142	155	24	1	68	340	50	1
170000	149	165	24	1	71	348	50	1
175000	149	180	29	1	73	365	55	2
180000	158	190	30	1	75	379	65	2
185000	166	197	31	1	76	385	102	3
190000	171	202	32	1	76	390	130	3
195000	171	210	32	1	77	399	315	3
200000	182	217	34	1	80	404	370	3
205000	192	228	35	1	80	410	424	3
210000	195	232	36	1	84	425	469	3

Table 4: Attack Types Detected per Flows Number — Execution 1

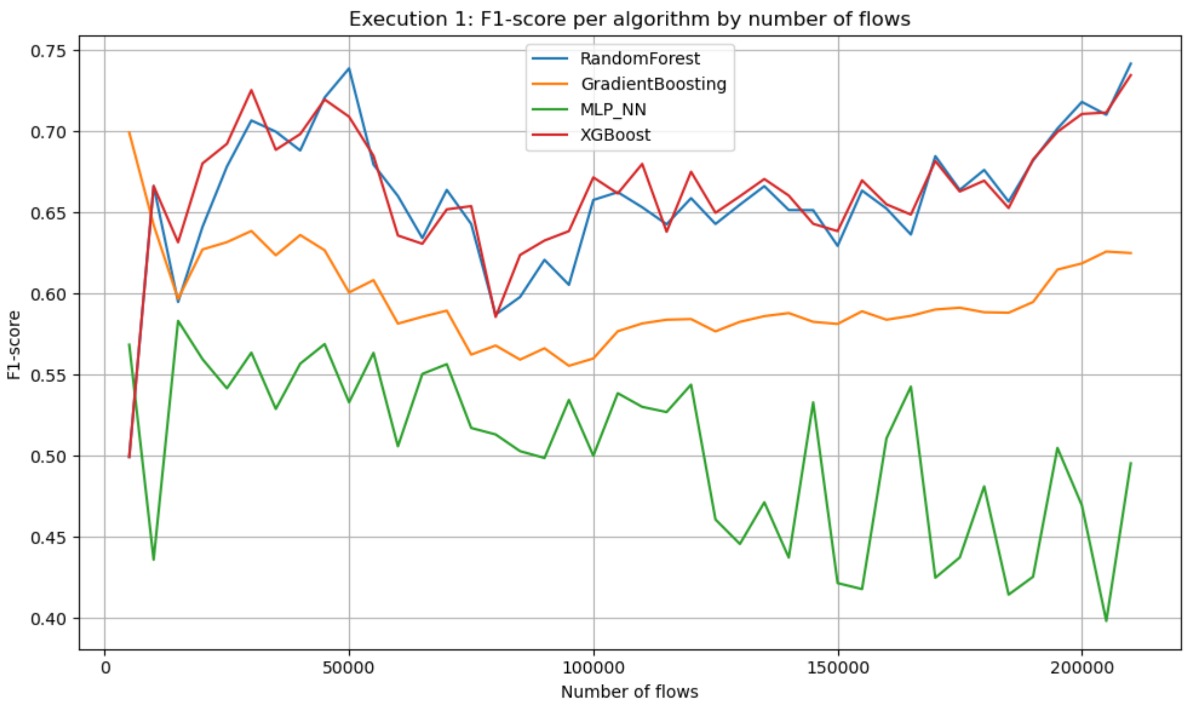


Figure 14: F1-score per algorithm by number of flows.

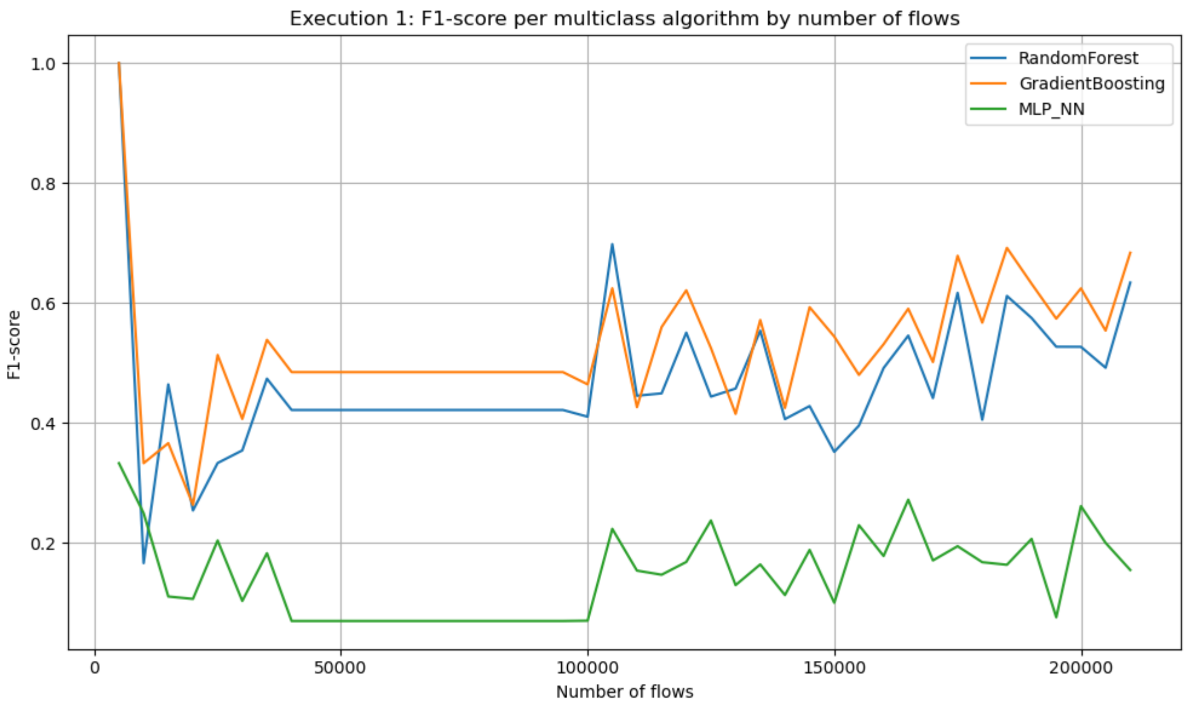


Figure 15: F1-score per multiclass algorithms by number of flows.

5.2 Execution 2

Number of Flows	Number of Attacks
5000	88
10000	184
15000	449
20000	590
25000	714

Table 5: Number of Attacks Detected per Flows Number — Execution 2

	Fuzzers	Reconnaissance	Shellcode	DoS	Exploits	Generic	Worms	Backdoor
5000	12	17	4	3	37	15	0	0
10000	27	25	8	6	50	67	1	0
15000	44	37	8	8	57	294	1	0
20000	66	56	11	12	78	366	1	0
25000	72	71	12	14	117	426	1	1

Table 6: Attack Types Detected per Flows Number — Execution 2

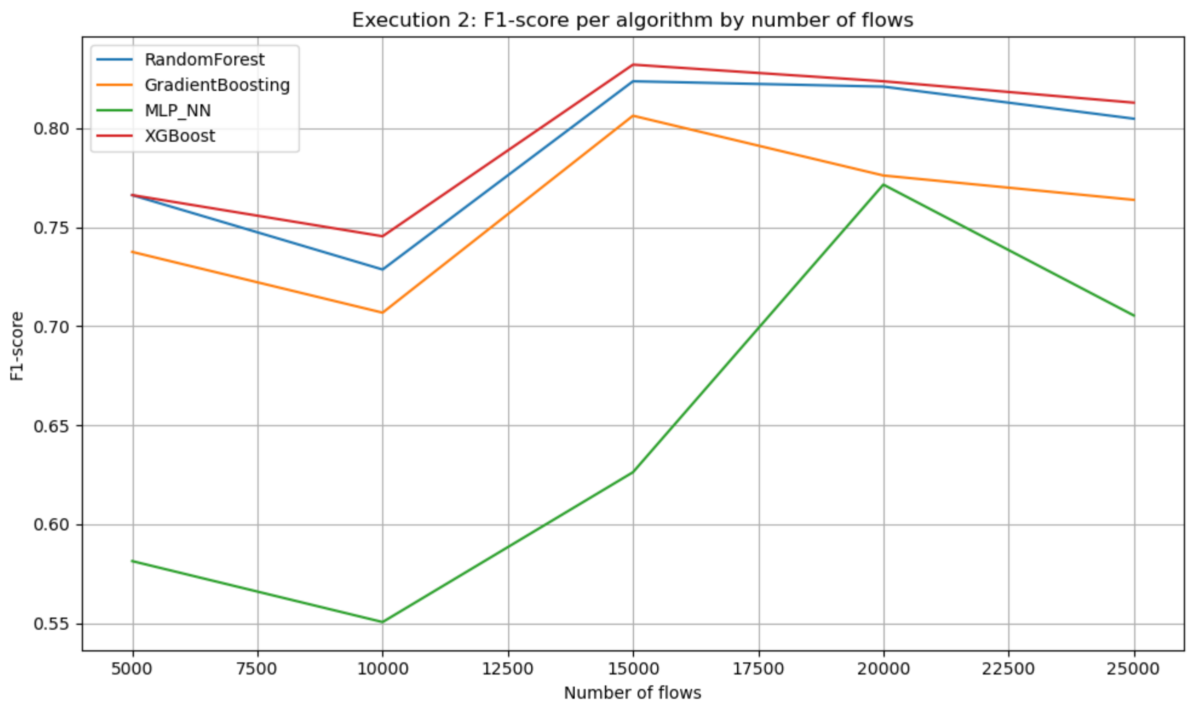


Figure 16: F1-score per algorithm by number of flows.

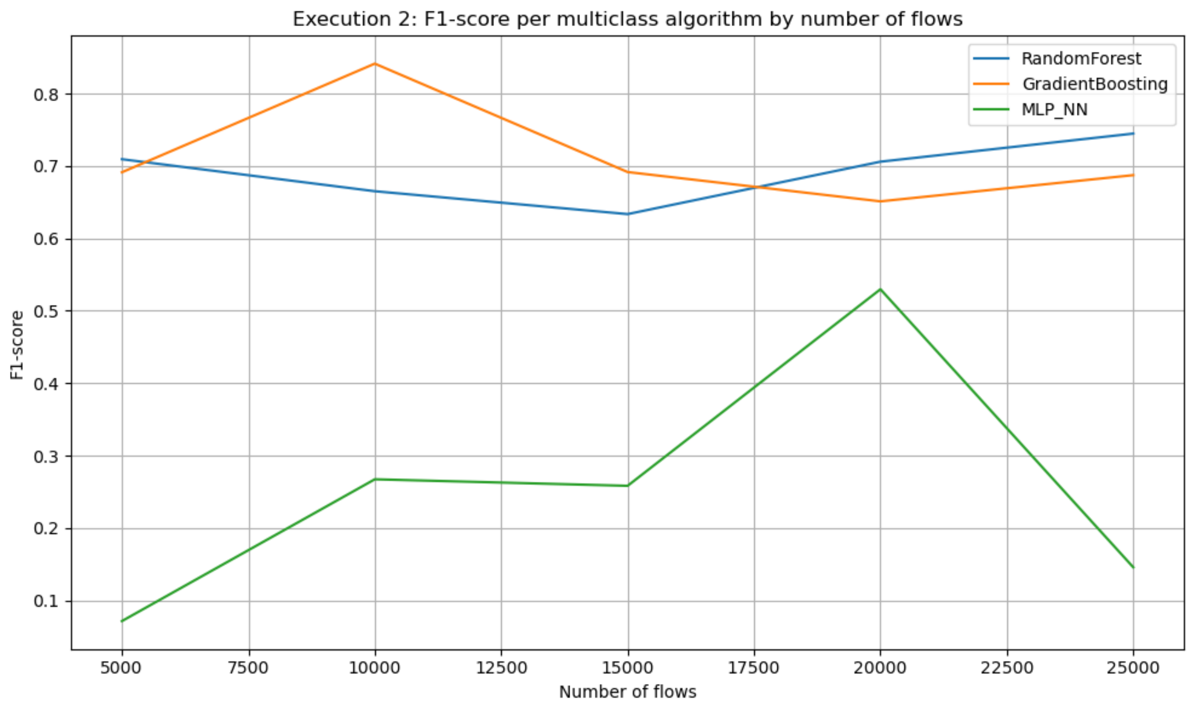


Figure 17: F1-score per multiclass algorithms by number of flows.

6 Discussion

This section analyzes the results obtained from our system in the previous section, highlighting both its strengths and the challenges encountered during development. The discussion provides context for understanding what influenced the results and outlines key considerations for future improvements.

6.1 MLOps Pipeline

The MLOps pipeline was designed to manage the entire ML lifecycle of the system, from data ingestion and preprocessing to model training, deployment, and evaluation. The pipeline was tailored to handle network traffic data and support dynamic updates in a near-real-time environment.

One of the first challenges we encountered involved handling streaming data in batches. During preprocessing, raw network traffic was split into fixed-size batches for processing. However, this approach sometimes caused individual flows to be fragmented across batch boundaries, resulting in incomplete or truncated flow records. These partial flows led to inaccurate feature extraction, ultimately degrading the performance of the classification models. To mitigate this issue, we adjusted the batch size to reduce the likelihood of cutting flows mid-transmission, thereby preserving the integrity of flow-based statistics and improving detection reliability.

Another issue we encountered was related to the early decision to convert raw network traffic into JSON format. This was initially motivated by the structure of NWDAF APIs, which are designed to consume and expose data using JSON. However, translating raw packet-level data into a consistent JSON representation proved to be highly problematic. Network traffic includes a wide range of protocols and packet structures, and maintaining a one-size-fits-all JSON schema became unfeasible. Moreover, the tools available for converting JSON back into a form suitable for NetFlow-based feature extraction, such as that required by nProbe, were either unreliable or nonexistent. Ultimately, this led us to abandon the JSON-based approach and work directly with raw network data, which ensured compatibility with the feature extraction tools and simplified the pipeline.

Regarding system performance, we evaluated both binary and multiclass classification tasks as the number of processed flows increased.

In the binary case (Figure 14), F1-scores improved quickly with the initial increase in data but later began to fluctuate. This behavior aligns with Table 3, which shows that in several intervals, the number of unique attack samples remained unchanged despite more flows being processed. This lack of new attack information likely contributed to the temporary stagnation or decline in model performance. As more diverse attacks were reintroduced later, performance began to rise again at higher levels.

In the multiclass case (Figure 15), results were more erratic, especially when attack diversity was low. As with the binary case, improvements became more consistent when new attack types were added to the dataset. Sampling techniques were useful during the early phases but had limited effect once the underlying class imbalance was no longer severe. Overall, the system benefited most when both the quantity and diversity of attack samples increased.

At this stage, model retraining is based solely on the availability of new labeled data. Although several other triggers (such as performance degradation, shifts in traffic patterns, or signs of concept drift) were considered, we chose to proceed with a data-availability-based approach, as it provided a more controlled setup to verify and validate the retraining process. This method proved sufficient for the scope and objectives of the current project, allowing us to demonstrate end-to-end retraining functionality within the pipeline.

In summary, the MLOps pipeline involved real-time data ingestion, feature extraction from raw Netflow, intelligent sampling techniques to handle imbalance, dynamic model comparison, and automatic deployment of the best model. A key achievement of our pipeline was the implementation of CI/CD mechanisms for trained models. This allowed the system to seamlessly integrate new models into the inference service with minimal downtime, enabling rapid iteration, adaptation to new data patterns, and improved maintainability over time. This setup enabled the system to stay responsive, adaptable, and scalable within a production-like environment.

6.2 5G-Core Integration

Integrating our pipeline with the 5GC network presented several challenges, primarily due to the complexity and breadth of the APIs defined by the 3GPP standards. These APIs are extensive, highly technical, and still evolving, which makes their interpretation and practical implementation particularly difficult. Additionally, many parameters specified within these interfaces are tailored for production-grade deployments. This limits their direct applicability in academic or prototyping environments.

Despite these challenges, our pipeline was designed with 5G integration in mind and already supports basic interaction patterns expected in such contexts. Through RESTful APIs developed using FastAPI, the system is capable of receiving network-related data and exposing processed results or ML inferences to external consumers. This enables simulated communication with 5G NFs, both as a data consumer and as a data provider, following the architectural principles outlined by the 3GPP.

One of the main limitations we encountered was related to the available infrastructure at the *Instituto de Telecomunicações* (IT). Although the IT provides access to a 5GC instance, the current deployment does not expose the necessary interfaces. Without these APIs, it was not possible to connect our system directly to real NFs like the AMF, SMF, or NRF.

Another limitation is found in the 3GPP TS 29.520 specification for ML model monitoring. While it defines a standardized interface for exposing model evaluation metrics, it only supports basic indicators such as accuracy. This is insufficient for production-ready analytics systems, where deeper insight into model behavior is critical. Metrics like precision, recall, F1-score, confidence intervals, or drift indicators are essential to effectively monitor and maintain ML models, especially in security-sensitive applications like anomaly detection. Future revisions of the system should incorporate an extended monitoring layer to supplement TS 29.520 with more detailed performance indicators.

To address this, we simulated the behavior of NFs and traffic using tools such as `Scapy`. The pipeline was built around a modular API layer that preserves compatibility with 3GPP principles. This ensures that, in future iterations, it can be integrated with more complete or open-source 5GC implementations such as Open5GS or Free5GC. With minimal adjustments, the system can be validated in realistic environments where actual traffic and network feedback are available.

Nevertheless, further testing in a real 5GC setup will be essential to fully validate the system. While the current proof-of-concept demonstrates the architectural viability and functional correctness of the pipeline, its performance, robustness, and adaptability under real-world operating conditions must be assessed. Such validation would involve end-to-end integration with live NFs, real network telemetry, and runtime orchestration logic, allowing us to evaluate aspects like data fidelity, inference latency, and automation responsiveness in realistic scenarios.

7 Future Work

The system developed in this project delivers a modular and functional MLOps pipeline for 5G network analytics, meeting the objectives defined for the selected use case. Its architecture supports real-time anomaly detection and provides a solid foundation for further development. Several improvements have already been identified to expand its scalability, integration with real environments, and long-term adaptability.

One important improvement is the migration from Docker Compose to Kubernetes. This change would enhance the system's scalability, resilience, and support for dynamic resource allocation.

Another area of development is the integration with real components of the 5GC, such as the AMF, SMF, UPF, and NRF. However, this integration was not performed during the current phase of the project because the 5GC system provided by the *Instituto de Telecomunicações*, based on Huawei's implementation, does not expose the necessary APIs for external access.

To build on this foundation and support more robust lifecycle management of ML components, a key next step is to improve visibility into model behavior. One planned enhancement is the implementation of a monitoring API compliant with the 3GPP TS 29.520 specification. This API would expose metrics such as confidence levels, prediction accuracy, and anomaly scores. It should also support automatic retraining when thresholds are exceeded, ensuring that model performance remains consistent over time and that operators are notified of any relevant changes.

Additionally, the system is expected to evolve towards interacting directly with other NFs when an attack is detected. For example, it could invoke the ReleaseUEContext operation via the `Namf_Communication` service [2], block malicious User Equipment (UE), or update policies via the PCF. These interactions would enable rapid and coordinated responses to threats in real time.

Overall, the system is well positioned for future expansion. Its current architecture supports key extensions that will improve automation, resilience, and integration, making it suitable for deployment in realistic and dynamic 5G network scenarios.

8 Conclusion

This project stands as a comprehensive exploration of how ML and modern software practices can be applied to enhance intelligence and automation within the evolving context of 5G networks. Anchored around the concept of NWDAF, our work aimed to simulate the kind of data-driven network intelligence envisioned in 3GPP architectures, where real-time analytics, automation, and adaptability are core pillars.

Though the specific use case chosen focused on real-time attack detection, the primary objective of the project was to validate broader architectural principles. We showed how complex, high-volume network data can be ingested, processed, analyzed, and acted upon dynamically within a distributed MLOps pipeline.

This system emulates what a true NWDAF-enabled pipeline might look like: handling real network traffic, extracting meaningful features from low-level flows, dynamically training and selecting the best-performing models, and delivering them through an automated, continuously updating inference service. In doing so, we addressed not only the technical challenges of ML lifecycle management, but also the broader systems integration concerns that come with deploying intelligence into real-time telecom environments.

Perhaps one of the most valuable achievements was the development of a pipeline capable of continuous delivery and deployment, ensuring that models are always up to date, automatically integrated, and ready to respond to new network conditions. Additionally, we implemented a training API, allowing network operators to trigger new training cycles and evaluate model performance periodically, providing greater control and visibility into the evolution of deployed models. This aligns with the ambition behind NWDAF: enabling mobile networks to become increasingly autonomous, self-optimizing, and responsive.

We also gained crucial insight into the limitations of current tooling, the importance of data quality and balance, and the difficulties of standardization in such a heterogeneous domain. These lessons, often hard-won, shaped a more resilient and pragmatic design philosophy as the project matured.

Ultimately, this project demonstrated how ML can be thoughtfully embedded into the operational fabric of future mobile networks. By aligning our implementation with NWDAF principles and 5G requirements, we delivered a functional and extensible platform that not only meets present demands but is also prepared to evolve. The combination of automation, observability, adaptability, and data-centric design gives this work lasting value as a practical reference for intelligent network analytics moving forward.

References

[1] 3GPP. 5g system overview. https://www.3gpp.org/technologies/5g-system-overview.

[2] 3GPP. 3gpp ts 29.518 - 5g system; access and mobility management services; stage 3. https://www.3gpp.org/ftp/Specs/archive/29_series/29.518/, 2024. Particularly the ReleaseUEContext operation exposed by the Namf_Communication service. See also: https://github.com/jdegre/5GC_APIs/blob/117674b61cd9ae4f68f5b5f91ec63686b8930cab/TS29518_Namf_Communication.yaml#L664.

[3] Rui Ferreira, João Fonseca, João Silva, Mayuri Tendulkar, Paulo Duarte, Marco Araújo, Raul Barbosa, Bruno Mendes, and Adriano Goes. Demo: Enhancing network performance based on 5g network function and slice load analysis. In *2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2023.

[4] Majed Luay, Siamak Layeghy, Seyedehfaezeh Hosseininoorbin, Mohanad Sarhan, Nour Moustafa, and Marius Portmann. Temporal analysis of netflow datasets for network intrusion detection systems. https://arxiv.org/html/2503.04404v1, 2025.

[5] Abdelkader Mekrache, Karim Boutiba, and Adlen Ksentini. Combining network data analytics function and machine learning for abnormal traffic detection in beyond 5g. In *GLOBECOM 2023 - IEEE Global Communications Conference*, 2023.

[6] Nour Moustafa and Jill Slay. Unsw-nb15 dataset. https://research.unsw.edu.au/projects/unsw-nb15-dataset, 2015.

[7] N. Nisha, K. Lakshman, and R. Kumar. A smart data analytics system generating for 5g n/w system via ml based algorithms for the better communications. In *2024 1st International Conference on Innovative Sustainable Technologies for Energy, Mechatronics, and Smart Systems (ISTEMS)*, 2024.

[8] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 27, 2021.