US PROJ 2023/2024
Advisor: Stéphane Rovedakis

Project # 4
NWDAF:
Network Data Analytics function

NWDAF: Demonstration of a Network Data Analytics function

Advisor: **Stéphane Rovedakis**

Student: **Fatemeh Salmani**

Table of Contents

Network Data analytics Function

1. Introduction:

The Network Data Analytics Function (NWDAF) emerges as a pivotal component within the 5G Core Network architecture, embodying the fusion of advanced artificial intelligence (AI) and machine learning (ML) technologies to extract actionable insights from network data. Defined by the Third Generation Partnership Project (3GPP), NWDAF serves dual roles: as a collector and analyser of network data, facilitating diverse applications from user mobility analytics to anomaly detection, and as a microservices architecture, enabling the integration of 3GPP applications as modular microservices for enhanced flexibility and scalability. NWDAF is implemented through two distinct approaches. The two current implementations include a C implementation by the Open Air Interface and Python version which both will be explained in detail below.

2. Based on OpenAirInterface:

https://www.eurecom.fr/en/publication/7410

2.1. OpenAirInterface NWDAF Introduction:

The project focuses on the integration of the Network Data Analytics Function (NWDAF) into the 5G Core Network (CN) architecture to address the challenges of accommodating end users and improving network performance. The NWDAF plays a crucial role in generating analytics and insights from network data, supporting various applications such as User Equipment (UE) mobility analytics and abnormal behavior detection.

To enhance the flexibility and scalability of NWDAF, the project proposes a microservices architecture, allowing the integration of 3GPP-defined applications as microservices. Additionally, a Machine Learning (ML) algorithm, specifically a Long-Short-Term-Memory (LSTM) Auto-encoder, is devised for detecting abnormal traffic events. The ML algorithm is trained using real network data from the Milano dataset.

The architecture consists of three layers: Exposure, Monitoring, and Analytics. The Exposure layer serves as the entry point for external entities, implementing a 3GPP-compliant interface. The Monitoring layer collects data from various sources, and the Analytics layer houses microservices engines responsible for different NWDAF services.

The project integrates the NWDAF with the OpenAirInterface (OAI) 5G CN, specifically with the AMF and SMF components. An abnormal traffic detection algorithm based on LSTM Auto-encoder is implemented as a microservice within the NWDAF architecture. Experimental results demonstrate the NWDAF's ability to collect data from a real 5G CN using 3GPP-compliant interfaces and detect abnormal traffic generated by a real UE using ML.

Use cases of NWDAF include core services such as network performance, UE communications, and NF

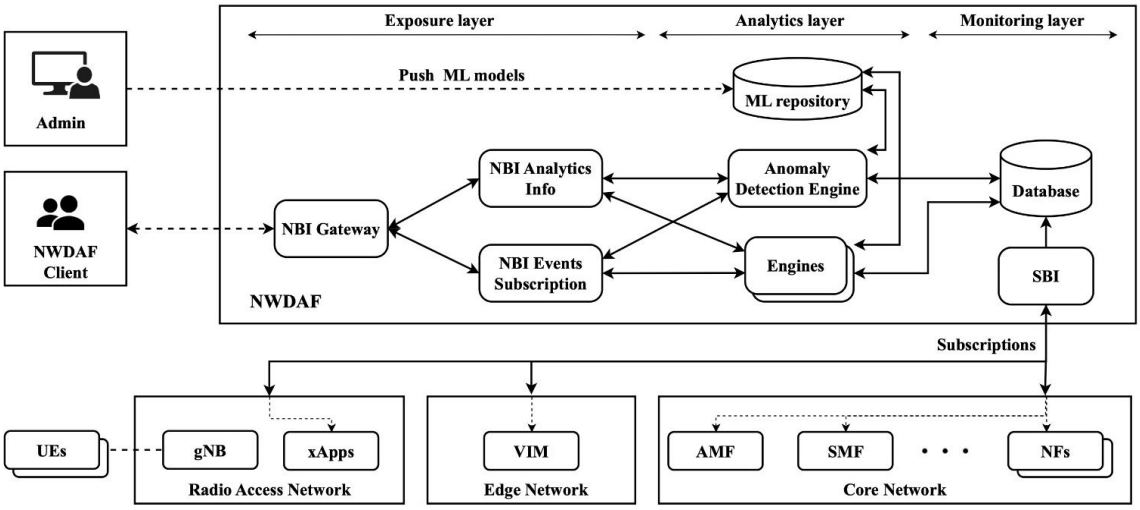
load, as well as ML-based services for abnormal traffic detection. The ML model's application extends to potential services like detecting Distributed Denial of Service (DDoS) attacks and predicting radio link failures.

The project's performance evaluation involves an experimental setup with two machines running the OAI 5G CN and NWDAF, along with a laptop representing a 5G UE. The LSTM Auto-encoder is trained on the Milano dataset, and the results show the model's ability to detect anomalies in network traffic.

In conclusion, the project contributes to the development of NWDAF and its applications in Beyond 5G networks, particularly in abnormal traffic detection. The proposed microservices architecture enhances flexibility, and the integration of ML algorithms demonstrates the potential for advanced use cases, ultimately aiming to improve network performance and end-user experience in the evolving landscape of mobile communication technologies.

2.2. NWDAF Microservices Architecture:

It introduces a microservices architecture for NWDAF, which allows plugging 3GPP applications as microservices, enhancing flexibility and scalability. The architecture consists of three layers: Exposure, Monitoring, and Analytics.



Exposure layer:

NWDAF Client: Clients can submit requests following 3GPP specifications. It can either request information or subscribe for an event. The client request is routed to either the NBI Analytics Info module or the NBI Events Subscription module, depending on the nature of the request.

NBI Gateway: North-Bound Interface (NBI) Gateway module, which acts as a 3GPP-compliant interface between NWDAF and its clients.

NBI Analytics Info: NBI Analytics Info module enables external entities to obtain real-time analytical data from the NWDAF, offering valuable insights into network performance. Its main responsibilities include: (i) handling requests from the NBI Gateway module; (ii) requesting the Analytics layer to compute the desired information; (iii) and generating and transmitting responses to clients based on the Analytics layer's output.

NBI Events Subscription: NBI Events Subscription module allows external entities to subscribe to specific network events and receive notifications when those events occur. This component performs several tasks, including: (i) handling requests from the NBI Gateway module; (ii) keeping track of existing subscriptions; (iii) communicating with the Analytics layer to compute the desired information; (iv) and delivering notification messages to the corresponding clients based on the responses received from the Analytics layer.

Analytics layer:

Within the architecture of NWDAF, each service is mapped to an engine located in the Analytics layer. Each engine performs the required computations to derive the target service information. The Exposure layer is responsible for selecting the appropriate engine based on the type of service requested.
For instance, The Anomaly Detection Engine employs an ML-based model to calculate the probability of traffic anomalies based on the history of the UE traffic patterns. The latter are retrieved from the database of the Monitoring layer by the engines. By comparing the current data pattern with the past, the engines can detect potential issues, and network administrators can take preventive measures.
The ML models are stored in a ML repository which is populated by the network administrator. The engines pull the ML model that corresponds to their proposed service.

Monitoring layer:

Monitoring layer is responsible for collecting data from different sources.

South-Bound Interface (SBI): It is a vital element responsible for collecting and storing network data. It communicates with several entities, such as NFs for CN-related data, VIM for Edge Network-related data, and xApps for RAN-related data. Upon startup of the NWDAF, the SBI module subscribes to the CN NFs to receive notifications and stores the received notification data in the database. Furthermore, it requests VIM to collect the RAM and CPU utilization of various NFs in the Edge network. For more details on the KPIs collection mechanism. The NWDAF can also request RAN information leveraging O-RAN xApps and KPM Service Model (SM).

2.3. NWDAF USE CASES:

NWDAF provides two main services: event subscriptions and analytics information. The NBI Events Subscription module enables clients to subscribe or unsubscribe to/from various analytics events, while the NBI Analytics Info module allows clients to request and receive specific types of analytics information. Our NWDAF provides both Core and ML-based services, which we will explore further.

2.3.1. Core services:

This proposed NWDAF supports three Core services:

Network Performance: The NWDAF provides support for two types of network performance events: “num of ue,” which measures the number of attach requests during a time window, and “sess succ rate,” which measures the session success rate during a time window specified in the request. The NWDAF computes “num of ue” using the AMF notifications, specifically, the registration event. The NWDAF also supports filtering the number of attach requests according to a specific network area or a specific operator. “sess succ rate” is computed using SMF notifications, i.e., the PDU session establishment event. The NWDAF also supports filtering according to a specific data network name or a network slice.

UE communications: “ue comm” refers to the number of packets and bytes exchanged in the uplink and downlink directions for each PDU session. To incorporate these statistics into SMF notifications, OAI-UPF-VPP2 was used during core network deployment. The SMF collects measurement reports from the UPF using the N4 interface for the usage report procedure of the N4 interface.

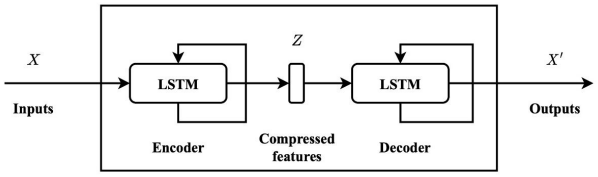
NF Load: As mentioned earlier, the NWDAF computes the “nf load” event data using data received from the VIM component. This data includes information on the CPU and RAM consumption of each NF.

Whenever a client subscribes for an event, the associated engine retrieves the required data from the MongoDB database and performs computation on the requested time window, i.e., the time interval of the client’s interest.

2.3.2. ML-based services

One of the key services of our NWDAF is ML-based abnormal traffic detection, which is identified in the 3GPP standard by the event ID “abnormal behaviour” and exception ID “unexpected large rate flow”.

Abnormal Traffic: The NWDAF clients can subscribe to the”abnormal behaviour” event to receive periodic updates on the probability of abnormal traffic. The latter is computed by the Anomaly Detection Engine leveraging an LSTM-based Auto-encoder. The LSTM-based Auto- encoder learns the traffic pattern using the history of the UE data measured from the UPF in both uplink and downlink directions. The traffic pattern contains information such as the data size in bytes at a given timestamp during a week.



An Auto-encoder is a type of artificial neural network that consists of an encoder and a decoder. During the encoding phase, the input data, represented as X = {x1 , x2 , . . . , xn }, is compressed into a lower-dimensional space according this Equation:

$$Z = \sigma\left(WX + b\right)$$

Where Z is the output of the LSTM encoder, σ is the activation function of the LSTM encoder, W is the weights of the encoder layers, and b is the bias vector of the encoder layers. Similarly, the decoding phase is trained according to bottom Equation to obtain the output data $X' = \{x'_1, x'_2, \ldots, x'_n\}$ that is similar to the input dimension.

$$X' = \sigma'\left(W'Z + b'\right)$$

Where Z is the input of the LSTM decoder, σ' is the activation function of the LSTM decoder, W' is the weights of the decoder layers, and b is the bias vector of the decoder layers. The Auto-encoder architecture's motivation is that it reduces the variance between input and output by training on only normal traffic without considering abnormal traffic. Whereas, the motivation behind combining LSTM and the Auto-encoder architecture is due to the nature of the input, which is correlated with time, e.g., traffic during the night is different from traffic during the day. Besides, the LSTM is well known for dealing with the vanishing gradient problem, which makes the training more stable. In bottom Equation , σ represents the Mean Absolute Error (MAE) between the input and output of the Auto-encoder.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |x'_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Let β denote the average MAE over the training data and β' the MAE of the inference data. Equation 4 calculates the traffic anomaly probability p using the difference between β and β'.

$$p = \min(\alpha \times |\beta' - \beta|, 1)$$

Where α is a weight to control the impact of the distances scale,i.e., when α is small enough, the distances will have less impact on the probability.

2.4. Technologies and programming languages:

The RESTful API architecture is used for all components due to its scalability, ability to manage multiple data formats, and utilization of HTTP.

Python is used for the development of engines that use ML models due to its extensive libraries and frameworks for data analysis and machine learning.

GoLang is used for the remaining system components due to its efficient code generation.

MongoDB is selected as the database due to its scalability and ability to manage unstructured data.

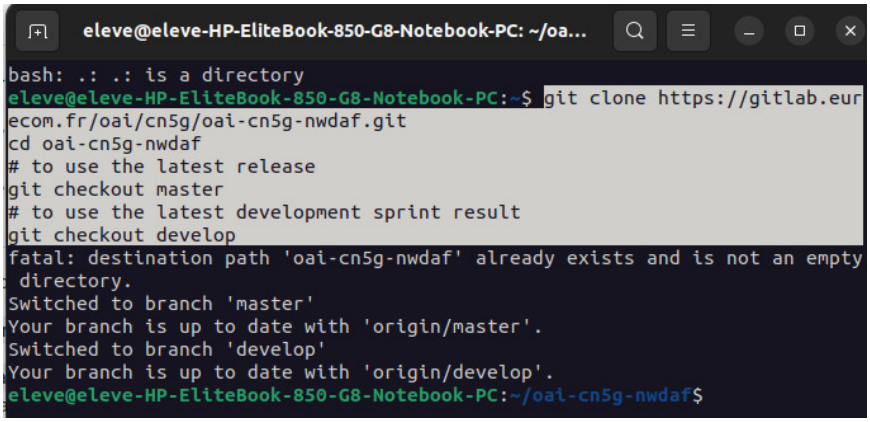
2.5. OpenAirInterface Implementation (Analytics part):

https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-nwdaf/-/blob/master/docs/TUTORIAL.md

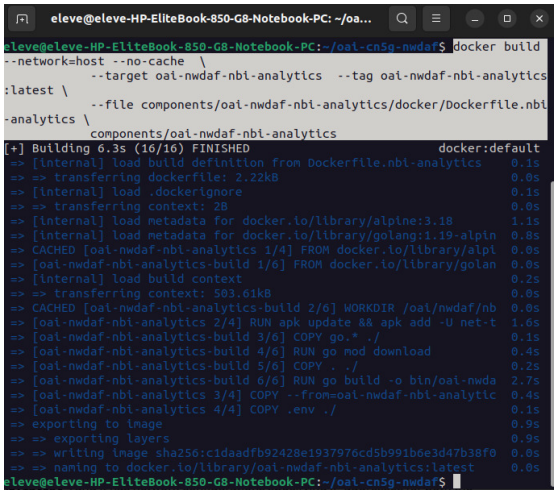
This section outlines the step-by-step process to set up and run the NWDAF project using container-based technology. Successful execution of these instructions is crucial for understanding the deployment and operational aspects of the NWDAF components.

2.5.1. Building NWDAF Images:

Clone the NWDAF repository from the provided source and navigate to the project directory.

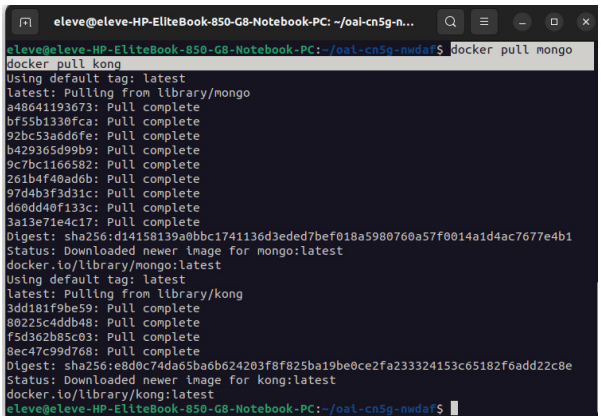


Build individual NWDAF Docker images using the specified Docker files and targets.

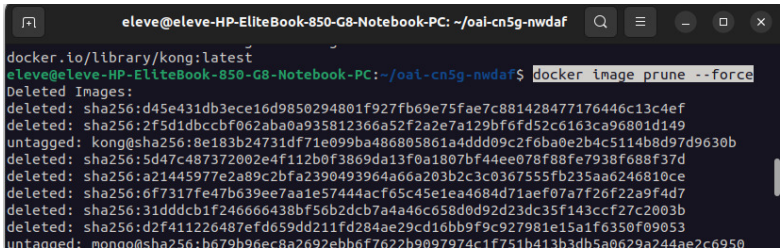


Repeat the above command for other NWDAF components (nbi-events, nbi-ml, engine, engine-ads, sbi).

Pull required images:

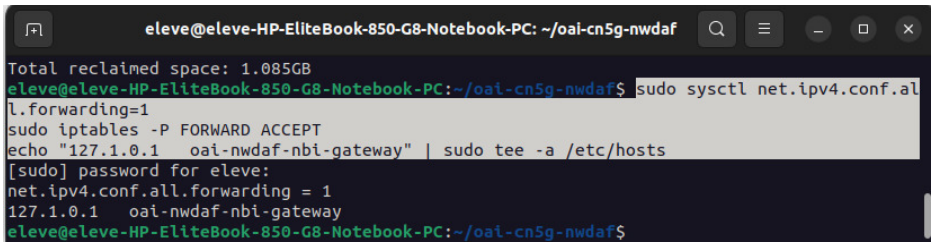


Remove dangling images:



2.5.2. Network Configuration:

Ensure a secure and functional network setup with the following commands:



Kong Gateway Setup:

- Docker Compose Configuration:

The project utilizes Docker Compose for orchestrating and configuring the necessary services. Two Docker Compose files, namely docker-compose-nwdaf-cn-http1.yaml and docker-compose-nwdaf-cn-http2.yaml, define the network services and their configurations.

- Kong Gateway Configuration:

Kong Gateway, a popular open-source API gateway, is employed in the project for managing API requests and interactions.

Docker Compose Service - oai-nwdaf-nbi-gateway:

Image: Kong:latest

Ports:

8000: Main port for API requests.

8443: Port for secure (HTTPS) API requests.

8001: Admin API port for configuration management.

Environment Variables:

KONG_DATABASE: "off" (Disabling Kong's database for declarative configuration.)

KONG_DECLARATIVE_CONFIG: Path to Kong's declarative configuration file.

KONG_PROXY_LISTEN: 0.0.0.0:8000 (Listening address and port for API requests.)

KONG_PROXY_LISTEN_SSL: 0.0.0.0:8443 (Listening address and port for secure API requests.)

KONG_ADMIN_LISTEN: 0.0.0.0:8001 (Listening address and port for Kong's Admin API.)

Volumes:

Declarative configuration file (kong.yml) is mounted to the container.

Networks:

Public and project-specific networks with assigned IP addresses.

- Kong Gateway Deployment:

The Kong Gateway deployment is configured using Kubernetes manifests (oai-nwdaf-nbi-gateway-deployment.yaml) within the Kubernetes namespace (oai-5g-core).

ConfigMap (kong-config):

Defines Kong's configuration, including services and routes.

Deployment:

Uses Kong image with specified ports and environment variables.

Mounts the configuration from the ConfigMap into the container.

Service:

Exposes the Kong Gateway service with ClusterIP, mapping ports 80, 443, and 8001.

- Service Configuration in Kong:

The Kong Gateway is configured to route and proxy requests to specific backend services:

oai-nwdaf-nbi-analytics

oai-nwdaf-nbi-events

oai-nwdaf-nbi-ml

Each service is associated with relevant routes, hosts, and paths to manage API endpoints effectively.

- Networking Considerations:

The project uses distinct networks for public access and internal project communication.

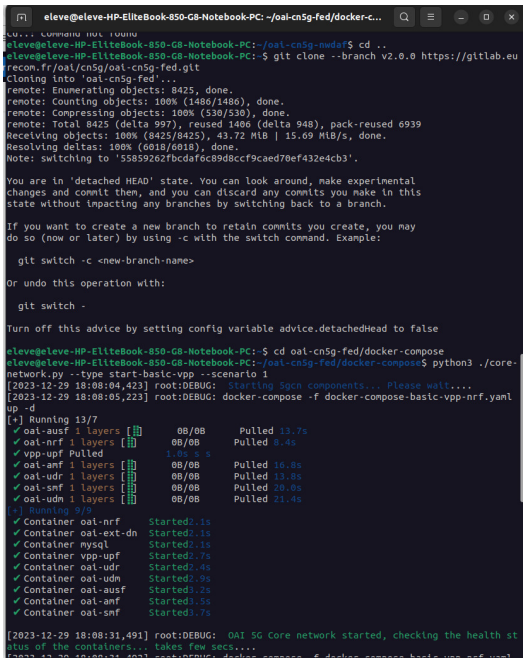
IP addresses are assigned to containers within the networks.

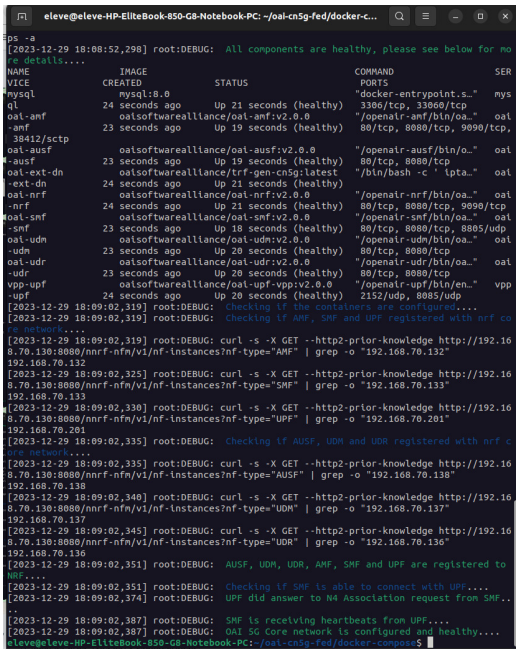
This network configuration, coupled with Kong Gateway, ensures efficient API management and secure communication within the project infrastructure.

2.5.3. NWDAF Deployment:

Starting 5G CN:

Clone the OAI 5G Core Network repository and deploy the OAI 5G Core Network:

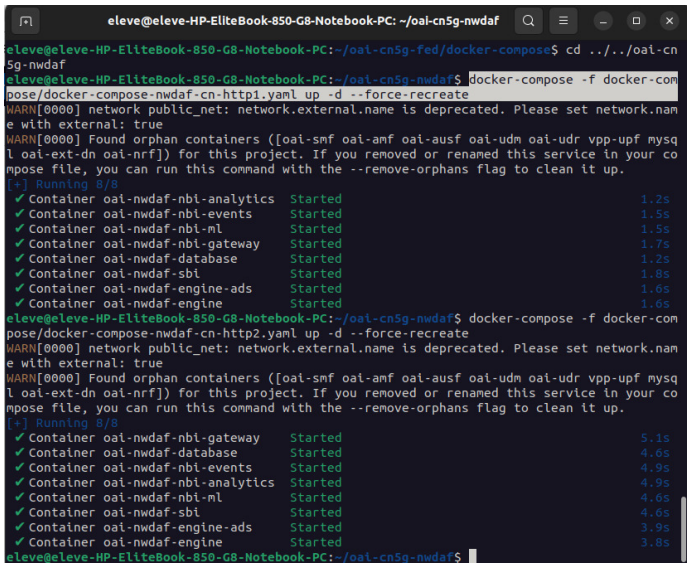




Warning: Restart NWDAF every time 5G CN is restarted to maintain correct behavior.

2.5.4. Starting NWDAF:

Navigate back to the NWDAF project repository and deploy NWDAF based on the AMF/SMF HTTP version:



Starting NWDAF Services with Docker Compose

The Network Data Analysis Function (NWDAF) services are orchestrated and deployed using Docker Compose, streamlining the initialization of the network functions. Two Docker Compose files, namely docker-compose-nwdaf-cn-http1.yaml and docker-compose-nwdaf-cn-http2.yaml, are utilized to define the services, configurations, and networking aspects.

Key Configurations:

- Container Configuration:

The services are defined with the oai-nwdaf-nbi-gateway container using the Kong image.

Essential environment variables are configured for Kong's functionality, including database settings and listening ports.

Configuration files (kong.yml) are mounted for declarative configuration.

- Networking Configuration:

Networking is established both within external networks (public_net) and an internal network (nwdaf_net) with specific IP addresses.

- Port Mapping:

Ports are mapped for API requests (8000), secure API requests (8443), and admin API (8001).

- Restart Policy:

A restart policy is set to "always" to ensure the services restart in case of failures.

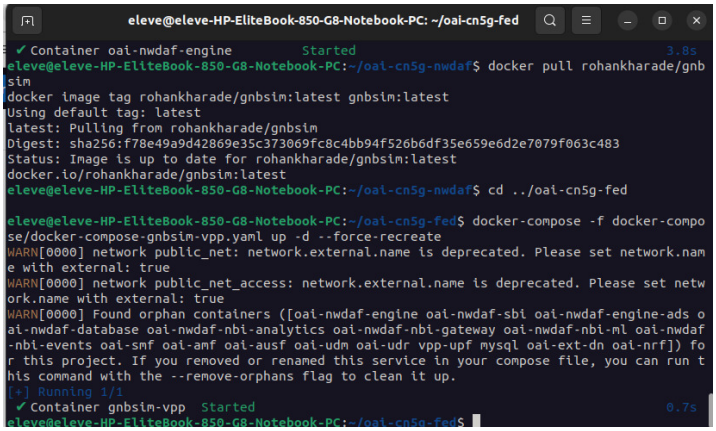
These Docker Compose configurations facilitate the seamless deployment of NWDAF services, providing a robust foundation for subsequent development and improvement.

2.5.5. Testing:

After ensuring both 5G CN and NWDAF are running, conduct various tests to validate NWDAF features.

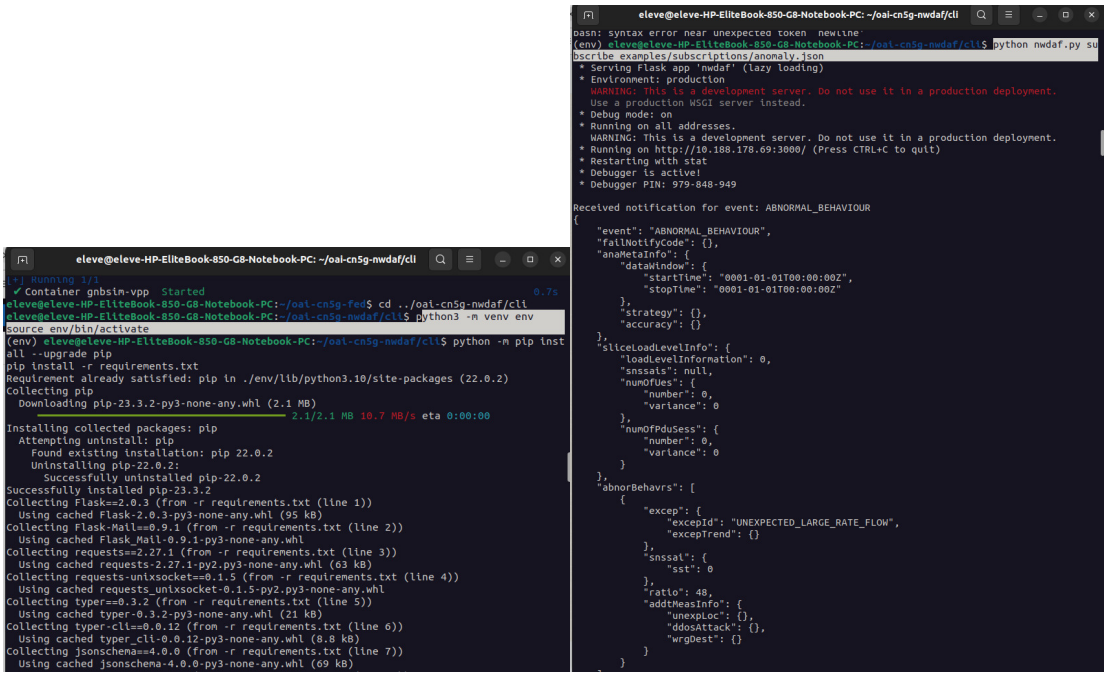
Attaching gnbsim:

Pull the gnbsim image and attach a UE using gnbsim-vpp:



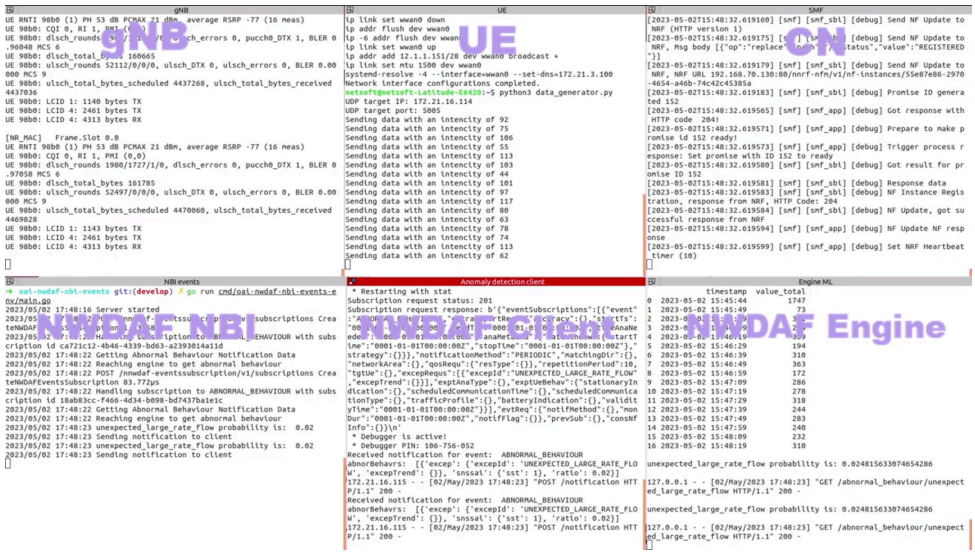
Testing NWDAF Features:

Run the provided CLI commands to test both the Analytics Info API and the Events Subscription API.



2.6. OpenAirInterface Implementation (ML- based):

https://www.youtube.com/watch?v=kI9GuJeW0es



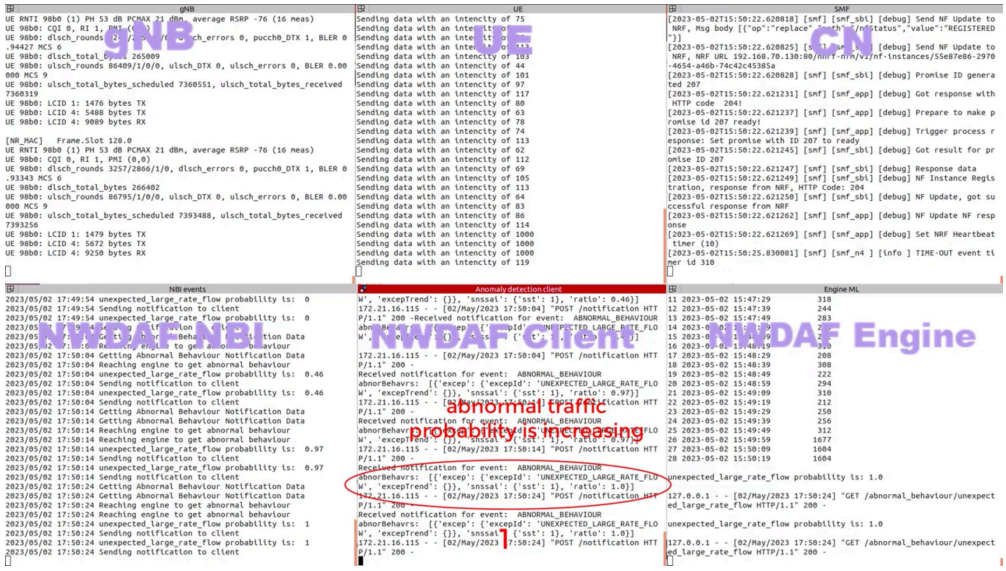
The experimental setup involves utilizing the OpenAirInterface (OAI) gNB, which is connected to the

USRP B210 and Quectel modem serving as User Equipment (UE).

The initiation of the process entails the generation of normal traffic patterns adhering to the Milano dataset, with the client subscribing to the NWDAF. Notably, under normal traffic conditions, the client experiences a negligible probability of receiving anomalous data. However, upon introduction of abnormal traffic, there is a discernible gradual increase in the probability of anomaly detection, eventually reaching a value of 1.

Upon detection, the client receives a notification, thereby enabling prompt identification and mitigation of traffic anomalies.

This methodology underscores a rigorous approach to anomaly detection and notification within the network environment, ensuring heightened awareness and proactive response to deviations from expected traffic behaviour.



3. Based on Python language:

3.1. Python NWDAF Introduction:

https://ieeexplore.ieee.org/document/9952559

This implementation provides analytic information to other network functions, application functions, and operation administration maintenance (OAM). The use cases of NWDAF have expanded to include quality of service (QoS) sustainability analytics, network performance analytics, and user equipment (UE) related analytics. NWDAF can be divided into two key components: Analytic Logical Function

(AnLF) and Model Training Logical Function (MTLF). AnLF is responsible for performing inference with ML models and providing analytics results to service consumers. MTLF is responsible for training ML models and providing ML model provisioning services. NWDAF communicates with other network components through service-based interfaces.

The NWDAF testbed was implemented to evaluate the functionality and performance of NWDAF. It consists of a Communication module and a Processing module. The Communication module uses Python Flask for handling HTTP messages with service consumers. The Processing module includes Controllers to identify HTTP request URIs and Models to determine what information should be returned.

3.2. Python NWDAF Architecture:

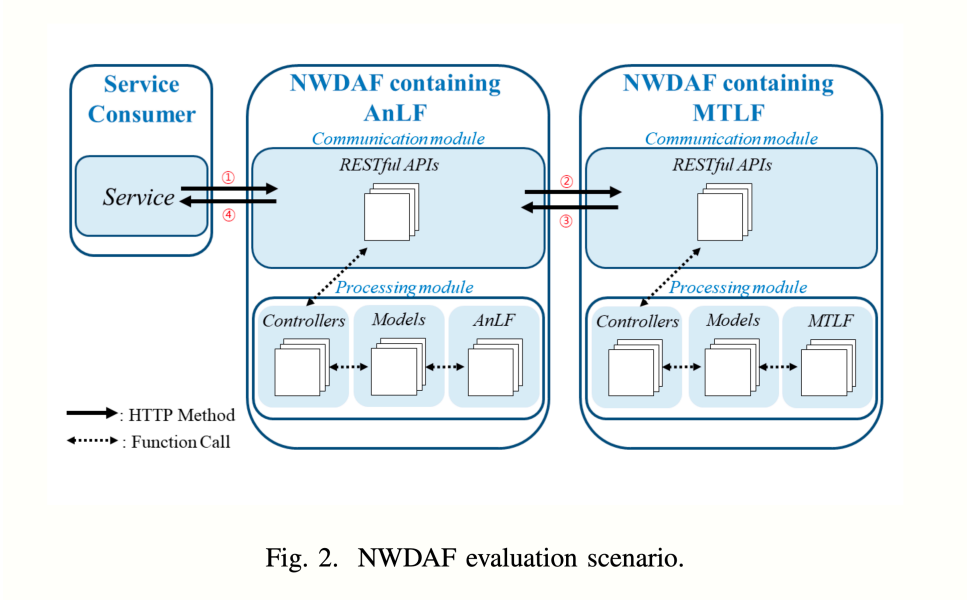


Fig. 2. NWDAF evaluation scenario.

The design of NWDAF Testbed comprises three main components: a service consumer, NWDAF containing AnLF, and NWDAF containing MTLF.

The service consumer sends an HTTP GET request to NWDAF containing AnLF to retrieve the NWDAF analytics. To request analytics information, the service consumer should include an event-id to identify which services to be analyzed. In this scenario, we set the event- id to" DATA ANALYTIC REQUEST" to see how it works.

When NWDAF containing AnLF receives the analytics re- quest through Communication module, Controllers find proper functions to serve the request in Models. NWDAF containing AnLF may need ML models to process the analytics services listed in the event-id. If AnLF has a proper ML model to perform inference, it immediately handles the request and returns the analytics results. However, if the proper ML model does not exist in AnLF, NWDAF containing AnLF sends an ML model provision request to NWDAF containing MTLF.

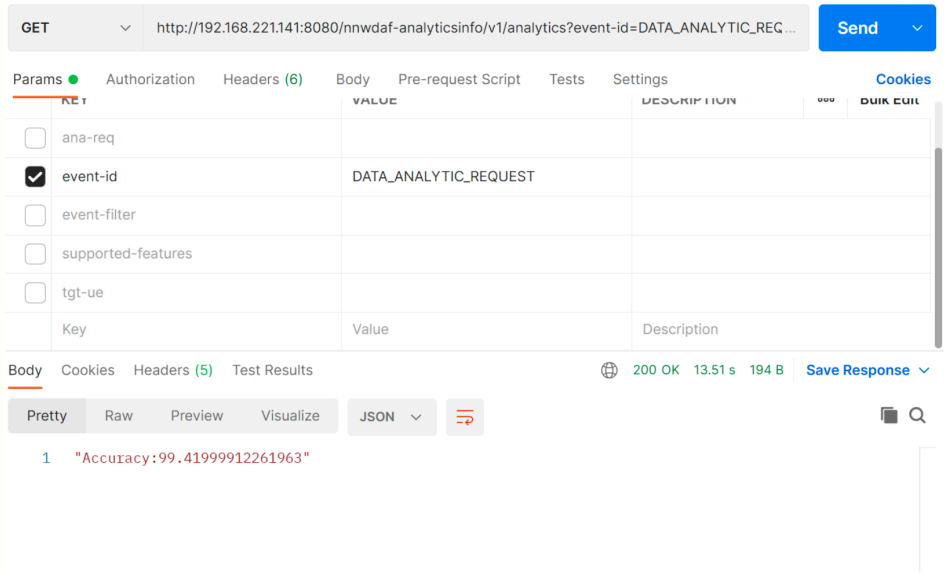
To request ML model provision, the service consumer sends an HTTP PUT request to subscribed NWDAF containing MTLF. When receiving the request, NWDAF containing MTLF responds with uniform resource locator (URL) or fully qualified domain name (FQDN) which indicates the address of the ML model file. This process is the same as that of NWDAF containing AnLF. After this procedure, NWDAF containing AnLF downloads the ML model file in the given URL and performs inference with the model and then returns the analytics results to the initial service consumer.

3.3. Python Implementation:

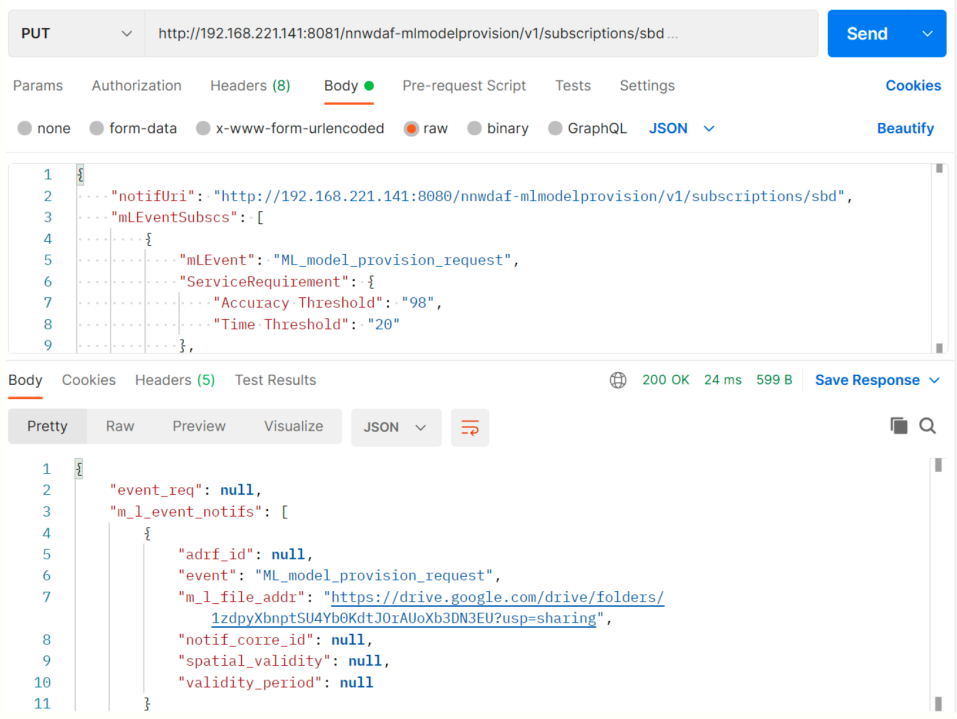
https://github.com/Lseung-hyun/NWDAF_Code

Analytics part: The implemented NWDAF provides analytics and ML model provisioning services through the NWDAF testbed. Below figure shows that the service consumer received the analytics results that match the requested event-id.

When NWDAF containing AnLF is requested by service consumers with the event-id for the first time, it cannot return the analytics results immediately. This is because AnLF cannot create ML models itself. So it does not have any ML model to process the analytics requests at first.



ML based: Below figure shows that NWDAF containing AnLF requests ML model provision to NWDAF containing MTLF. Right after receiving the ML model file address, AnLF downloads the ML model and performs inference with them. Meanwhile, when the service consumers send the same analytics request next time, NWDAF containing AnLF performs inference with the ML model that it already had.



4. Comparing two Implementation:

4.1. Introduction and Purpose:

OpenAirInterface NWDAF: The OpenAir NWDAF is designed to integrate seamlessly into the 5G Core Network (CN) architecture, primarily focusing on enhancing network performance and accommodating end users through analytics and insights generation. It emphasizes a microservices architecture for scalability and flexibility, with a particular emphasis on anomaly detection using Machine Learning (ML) algorithms.

Python NWDAF: The Python NWDAF serves a similar purpose, providing analytic information to various network functions and maintenance operations. It extends its capabilities to include quality of service (QoS) sustainability analytics and UE-related analytics. It comprises Analytic Logical Functions (AnLF) for inference with ML models and Model Training Logical Functions (MTLF) for ML model training and provisioning.

4.2. Architecture:

OpenAir NWDAF: It follows a three-layered architecture comprising Exposure, Monitoring, and Analytics layers. The Exposure layer handles external requests via a 3GPP-compliant interface, while the Monitoring layer collects data, and the Analytics layer hosts microservices engines responsible for different NWDAF services.

Python NWDAF: The Python implementation also adopts a modular approach, with a Communication

module handling HTTP messages and a Processing module managing request routing and analytics processing. It interacts with service consumers through HTTP requests and utilizes controllers and models for handling analytics tasks.

4.3. Technologies Used:

OpenAir NWDAF: It leverages technologies like RESTful APIs, Python for ML-based engines, GoLang for system components, and MongoDB for database management.

Python NWDAF: It primarily utilizes RESTful APIs and Python for both communication and processing modules, along with Flask for handling HTTP messages.

4.4. Experimental Setup and Evaluation:

OpenAir NWDAF: It integrates with the OAI 5G CN, specifically with components like AMF and SMF, and utilizes real network data for ML model training. Performance evaluation involves experimental setups with OAI 5G CN, NWDAF, and UE, demonstrating anomaly detection capabilities.

Python NWDAF: It implements a testbed to evaluate functionality and performance, consisting of communication and processing modules. The testbed allows for HTTP request handling and analytics processing, showcasing the interaction between AnLF and MTLF.

Overall, both versions of NWDAF share similar objectives and functionalities and they are differ in their implementation details, technology stack, and experimental setups.

5. References:

https://ieeexplore.ieee.org/document/9952559
https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-nwdaf
https://github.com/Lseung-hyun/NWDAF_Code
https://openairinterface.org/legal/oai-public-license/
https://www.etsi.org/deliver/etsi_ts/123200_123299/123288/16.04.00_60/ts_123288v160400p.pdf
https://the5gzone.com/index.php/nwdaf-introducing-machine-learning-capabilities-in-5g/
https://www.eurecom.fr/en/publication/7410
https://www.youtube.com/watch?v=kI9GuJeW0es