Grant Agreement No.: 101096342
Call: HORIZON-JU-SNS-2022
Topic: HORIZON-JU-SNS-2022-STREAM-B-01-04
Type of action: HORIZON-JU-RIA

Holistic, omnipresent, resilient services
for future 6G wireless and computing ecosystems

D3.2 – HORSE Platform Intelligence developed (IT-2)

Work package	WP 3
Task	Tasks 3.1, task 3.2, task 3.3, task 3.4 and task 3.5.
Due date	30/06/2025
Submission date	30/06/2025
Deliverable lead	ETI
Version	1.0
Authors	Orazio Toscano (ETI), Alessio Formica (ETI), Jose Manuel Manjón (TID), Juan Tamboleo (UMU), Fabrizio Granelli (CNIT), Malak Qaisi (CNIT), Eva Rodríguez (UPC), Panagiotis Gkonis (NKUA), Alice Piemonti (MAR), Vito Cianchini (MAR), Theodoros Velmachos (SUITE5) and Stefanos Venios (SUITE5)
Reviewers	Panagiotis Gkonis (NKUA), Alexandros Katsarakis (STS)

Abstract	This deliverable presents the final development of the modules of the Platform Intelligence., in terms of software descriptions and technical details.
Keywords	Development, implementation, configuration, installation.

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
V 0.1	02/05/2025	Table of Contents	Orazio Toscano (ETI)
V 0.2	30/05/2025	Introduction	Orazio Toscano (ETI)
	30/05/2025	Contribution to section 2.1	Jose Manuel Manjón (TID)
	30/05/2025	Contribution to section 2.1	Juan Tamboleo (UMU)
	30/05/2025	Contribution to section 4	Panagiotis Gkonis, Nikolaos Nomikos, Gerasimos Patsourakis, Vasileios Nikolakakis, Panagiotis Trakadas (NKUA)
	30/05/2025	Contribution to section 2.2	Fabrizio Granelli and Malak Qaisi (CNIT)
	30/05/2025	Contribution to section 5	Stefanos Venio (Suite5)
	30/05/2025	Contribution to section 6	Orazio Toscano (ETI)
	30/05/2025	Contribution to section 3	Eva Rodríguez (UPC)
V 0.3	02/06/2025	Modifications and comments	Orazio Toscano (ETI)
V 0.4	05/06/2025	Modifications and comments	All involved partners
V 0.5	20/06/2025	Comments from reviewers	Panagiotis Gkonis (NKUA) and Alexandros Katsarakis (STS)
V 1.0	30/06/2025	Final reviewed version	All involved partners

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	OTHER	
Dissemination Level		
PU	*Public, fully open, e.g. web*	**X**
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* *R: Document, report (excluding the periodic and final reports)*
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
DATA: Data sets, microdata, etc
DMP: Data management plan
ETHICS: Deliverables related to ethics issues.
SECURITY: Deliverables related to security issues
OTHER: Software, technical diagram, algorithms, models, etc.

Executive summary

This deliverable presents the final version of the software modules developed within Work Package 3 (HORSE Intelligence) of the HORSE research project. It marks the completion of the five key tasks composing WP3 and reflects the culmination of technical efforts carried out over the past 30 months of the project’s 36-month duration. This final release succeeds and builds upon the intermediate version delivered over a year ago, which focused on providing stable prototypes of the Platform Intelligence (PIL) components for initial integration (IT-1) into the HORSE architecture.

The modules addressed in this report are:

Sandboxing (SAN) – hosting the Digital Twins for scenario testing and predictive analysis,

Early Modelling (EM) – responsible for generating preliminary assessments through policies and rules,

Distributed and Trustable AI Engine (DTE) – ensuring secure and privacy-compliant AI data handling, threat identification and mitigation,

Policies and Data Governance (PAG) – managing and enforcing data policies across the platform, and

Threat Detector and Mitigation Engine (DEME) – focusing on detecting and mitigating security threats based on network behaviours and data analysis.

This final version documents the complete development, refinement, debugging, and internal validation of the above modules, collectively constituting the Platform Intelligence components at Integration Target 2 (IT-2). These components are now ready for full integration into the HORSE platform through Work Package 5 (Platform Integration, Use Case Deployment, Validation, and Final Release), which dominates the project’s final six months.

Importantly, the debugging and improvements in this version were informed by preliminary integration and validation activities already initiated under WP5. An internal technical report produced at Month 24 supported these early efforts (Task 5.2), and this final deliverable aligns with the planned conclusion of WP3 at Month 30.

With this comprehensive and stabilized release, the Platform Intelligence layer is fully prepared for its role in the final integrated platform. It ensures interoperability with the components from WP4 and sets a solid foundation for the final deployment and validation stages ahead.

Table of contents

List of figures

Abbreviations

5G	(Mobile) Fifth Generation
5GTN	5G Test Network
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
CPU	Central Processing Unit
DDoS	Distributed Denial-of-Service
DEME	Detector and Mitigation Engine
DN	Data Network
DNS	Domain Name System
DT	Digital Twin
DTE	Distributed and Trustable AI Engine
EM	Early Modelling
FL	Federated Learning
ENISA	European Network and Information Security Agency
gNB	Next Generation Node B
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IBI	Intent-Based Interface
IMEI	International Mobile Equipment Identity
IP	Internet Protocol
JSON	JavaScript Object Notation
KNE	Kubernetes Network Emulator

MEC	Multi-access edge computing
ML	Machine Learning
MSPL	Microsoft SIP Processing Language
NDT	Network Digital Twin
NEF	Network Exposure Function
NIDD	Network Intrusion Detection Dataset
NSA	Non-Standalone
NTP	Network Time Protocol
NWDAF	NetWork Data Analytic Function
P&P	Point-to-Point
PAG	Policies and Data Governance
PCAP	Packet Capture
PIL	Platform Intelligence
QoS	Quality of Service
RAN	Radio Access Network
REST	Representational State Transfer
SA	Standalone
SDN	Software-Defined Networking
SM	Smart Monitoring
SMF	Session Management Function
STO	Secure and Trustable Orchestration
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TTP	Tactics, Techniques, and Procedures

UDP	User Datagram Protocol
UERANSIM	UE and RAN simulator
UPF	User Plane Function
VM	Virtual Machine
XML	eXtensible Markup Language
YAML	Yet another markup language

1 Introduction

This deliverable presents the final outcomes of WP3 – Platform Intelligence within the HORSE project. As the second and concluding version of this document, it builds upon the initial results delivered during the first iteration (IT-1), and now reflects the completed implementation, refinement, and validation of all components associated with WP3.

Since this is categorized as an "OTHER" type deliverable, its focus remains primarily on the software development aspects of the involved modules, as well as their readiness for integration within the broader HORSE platform.

The document is organized into five main sections, each corresponding to a specific task within WP3:

- Section 2: Sandboxing, corresponding to Task 3.1
- Section 3: Early Modelling Framework, corresponding to Task 3.2
- Section 4: Distributed and Trustable AI Engine, corresponding to Task 3.3
- Section 5: Policies and Data Governance, corresponding to Task 3.4
- Section 6: Threat Detector and Mitigation Engine, corresponding to Task 3.5

Section 2 covers the final development of the Sandboxing (SAN) module, which includes two Digital Twin subcomponents: the Prediction and Prevention DT and the Impact Analysis DT. Together, they enable simulation and evaluation of various configurations and scenarios, offering a dynamic environment for testing platform behaviour before deployment.

Section 3 focuses on the Early Modelling (EM) component, which provides foundational input to the sandbox through two key elements: the Taxonomy, responsible for profiling and classifying system components, and the Attributes block, which defines the strategic criteria used to characterize modules based on specific parameters.

Section 4 describes the Distributed and Trustable AI Engine (DTE), which collects and processes data from multiple sources using machine learning and AI techniques to derive security policies while ensuring data privacy. It also supports data pre-processing for model training and integrates with other intelligence components in the platform.

Section 5 details the Policies and Data Governance (PAG) module, which functions as a central authority for data management across the platform. It ensures compliance with privacy, quality, and access requirements, while aligning with regulatory and ethical standards.

Finally, Section 6 presents the development of the Threat Detector and Mitigation Engine (DEME), designed to operate in complex and distributed environments. This module employs specialized algorithms to analyze network parameters, protocol headers, and real-time traffic from various network elements and virtualized functions (VNFs), with the goal of detecting and mitigating potential threats.

This final deliverable reflects the maturity of all WP3 modules, which are now stable and ready to be integrated into the full HORSE platform as part of the ongoing activities in WP5.

2 Development of the Sandboxing

2.1 Impact Analysis Digital Twin

The Impact Analysis Digital Twin has been deployed in Kubernetes [1] using mainly Kubernetes Network Emulator (KNE) [2]. The topology has been updated in the iteration 2 and the new one is shown in the following Figure 1:

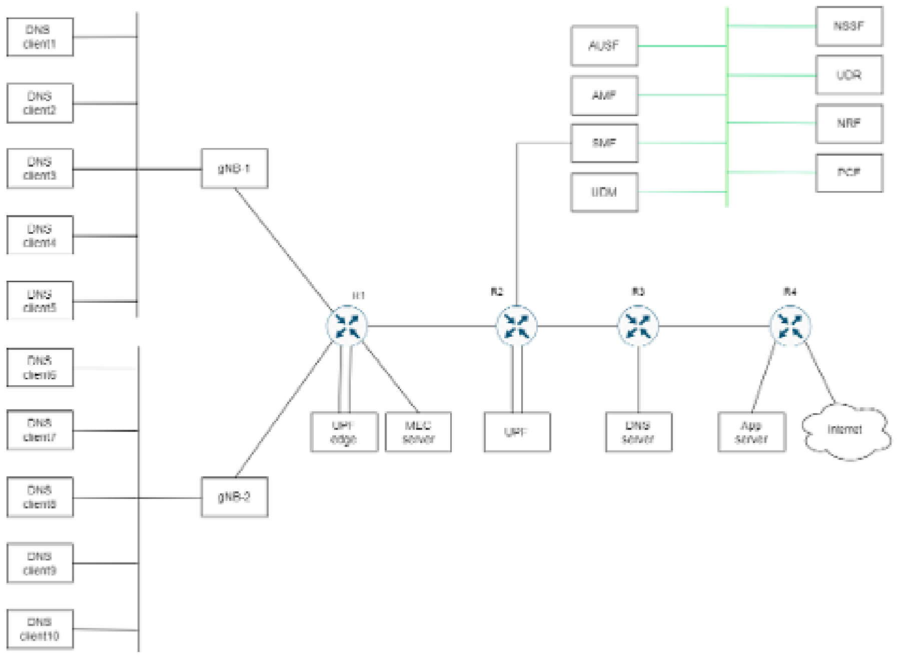


Figure 1: Network topology for iteration 2.

In this new network topology, we have included two gNBs, four routers, two UPFs, a MEC server, a DNS server and an Application Server jointly with the DNS clients on the 5G Core, deployed with Open5GS [3][3], [4].

The Impact Analysis Network Digital Twin has been improved by adding some automation mechanisms (using Ansible scripts) to make the deployment easier and faster.

The general architecture of the Impact Analysis Digital Twin is:

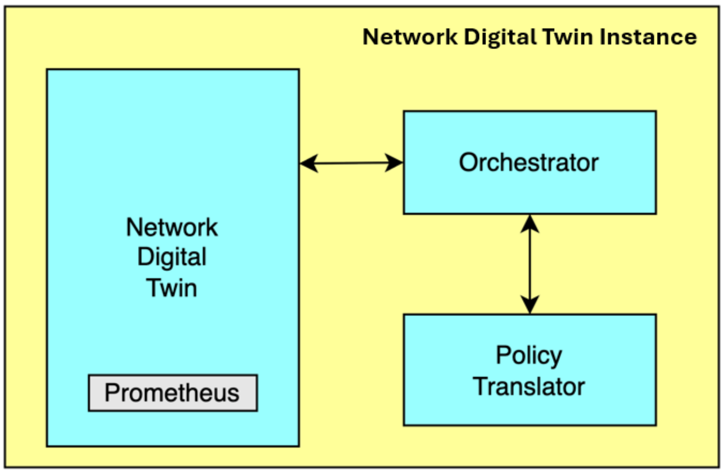


Figure 2: Architecture of the Impact Analysis Digital Twin.

In Figure 2 we can see that the IA-NDT is made up of different modules:

Network Digital Twin

Is the module that has the pods itself representing the different blocks of the topology previously described. The deployment of the Kubernetes cluster has been made using different dependencies such as Golang, Docker, kubectl, Kind and KNE in a Linux server.

Here, also is deployed a Prometheus [35] instance to get the different metrics from the NDT.

Orchestrator

UMU's security orchestrator[1] enables mitigation policies to be applied to an infrastructure, whether physical or virtual. Underneath, it is composed of plugins for translation and device-specific drivers that execute the action.

The policy format is MSPL, a cybersecurity policy-oriented XML. These policies have different purposes, such as applying Filtering, QoS, both on routers and hosts, modifying service configurations such as DNS or NTP, etc.

Policy Translator

The Policy Translator is a translation module that acts as an intermediate point between the IBI (for applying mitigations) or EM (for simulating attacks) and the orchestrator. Its function is to translate the outputs of these HORSE components into a valid format for the orchestrator, MSPL. It also allows setting timers and managing the creation/deletion of policies.

2.1.1 Interfaces

Regarding the modules that communicates with the Impact Analysis Digital Twin, it has different interfaces, all of them managed via REST API:

- **Intent-Based Interface (IBI)**: manages the what-if scenarios by sending the metrics that the NDT must monitor and send back the specific measures.

[1] *For more details about the UMU testbed please refer to D5.2*

- **Early Modelling (EM):** informs the NDT about the specifications of the attack and scenario to deploy it.

- **Smart Monitoring (SM)**: sends the information about the topology that will be used for the different scenarios.

2.2 Prediction and Prevention Digital Twin

The Prediction and Prevention Digital Twin is built on the Comnetsemu network emulation software [10], [11]. Comnetsemu is based on the well-known mininet network emulator [12], with the integration of a docker-in-docker environment to enable the deployment of services as docker containers. In this way, it is possible to emulate a 5G SA or NSA architecture by exploiting the available open-source implementations of the 5G core and access networks.

All employed software, including Comnetsemu, is publicly available and open source.

Mininet is a well-recognized Software Defined Networking network emulator. It is characterized by a stable and realistic performance, as demonstrated in [13], as well as some limitations in extremely large emulation scenarios [14]. Comnetsemu builds up on top of such realistic network emulation to enable to deploy actual service containers, thus generating a realistic workload and enabling to build realistic scenarios for current and next-generation networks.

Prediction and Prevention Digital Twin includes the following modules:

- Digital Twin Modelling module: it is responsible for generating the DT based on the input data (traffic and topology information, orchestrated services, etc.)
- Digital Twin Engine module: it will run the DT in the Comnetsemu emulation environment.
- Digital Twin-based Prediction module: it will analyze the output of the DT Engine block using AI/ML algorithms to perform predictions and identify anomalies.
- I/O Interface module: interface with DTE / IBI for receiving requests and providing the related outcomes.

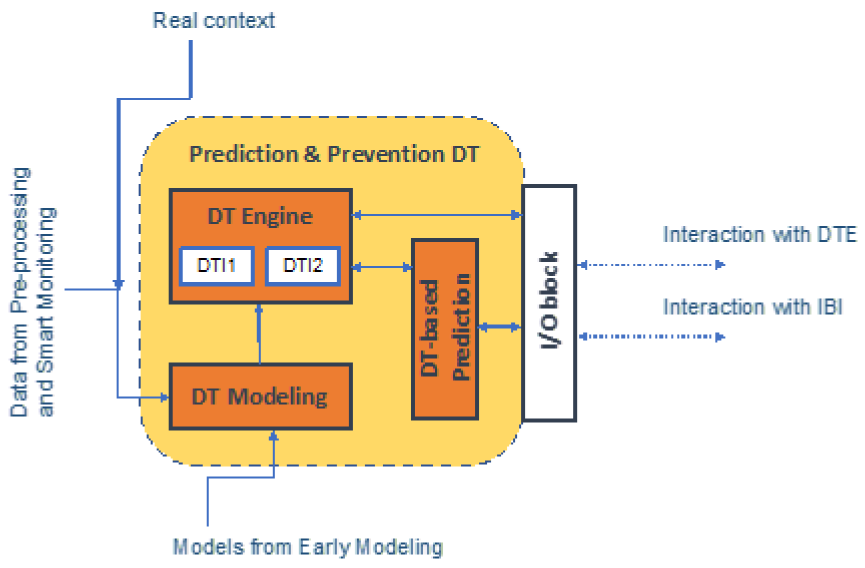


Figure 3: The structure of the Prediction and Prevention Digital Twin (from HORSE D2.2).

The Prediction and Prevention Digital Twin is available to the HORSE platform as a Virtual Machine. Deployment of the VM is performed through Vagrant.

All modules are developed in Python. The following sections describe how the internal modules are developed.

Digital Twin Modelling module

This module generates the script for replicating the Physical Twin of the 6G network into the Digital Twin.

This module receives in input via REST APIs a YAML descriptor of the network topology and the known services running in the network. The format for data collection is common for the entire HORSE sandbox, and it is the same as for the Impact Analysis Digital Twin. An example of the format of the file is as described in Section 2.1.2. Based on such information, it generates a script file to build the network and services in the Comnetsemu environment and to run the Digital Twin in the sandbox.

The following represents an example of a script for deploying a simple topology in mininet or Comnetsemu:

```
from mininet.topo import Topo

class MyFirstTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        leftSwitch = self.addSwitch( 's1' )
        rightSwitch = self.addSwitch( 's2' )
        # Add links
        self.addLink( h1, leftSwitch )
        self.addLink( h2, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, h3 )
        self.addLink( rightSwitch, h4 )

topos = { 'myfirsttopo': ( lambda: MyFirstTopo() ) }
```

The module receives as input via REST APIs also a YAML/XML descriptor of the attack or scenario to evaluate. This will be translated into a set of commands to deploy additional components in the Digital Twin and run/replicate network traffic.

Digital Twin Engine module

This module implements the Digital Twin. The Digital Twin is built in the Comnetsemu environment, enabling a precise emulation of an SDN network and faithful replication of services by deploying them in docker containers.

As an example, the following Figure 4 represents how a simple 5G network with Mobile Edge technology can be replicated in form of a Digital Twin in Comnetsemu.

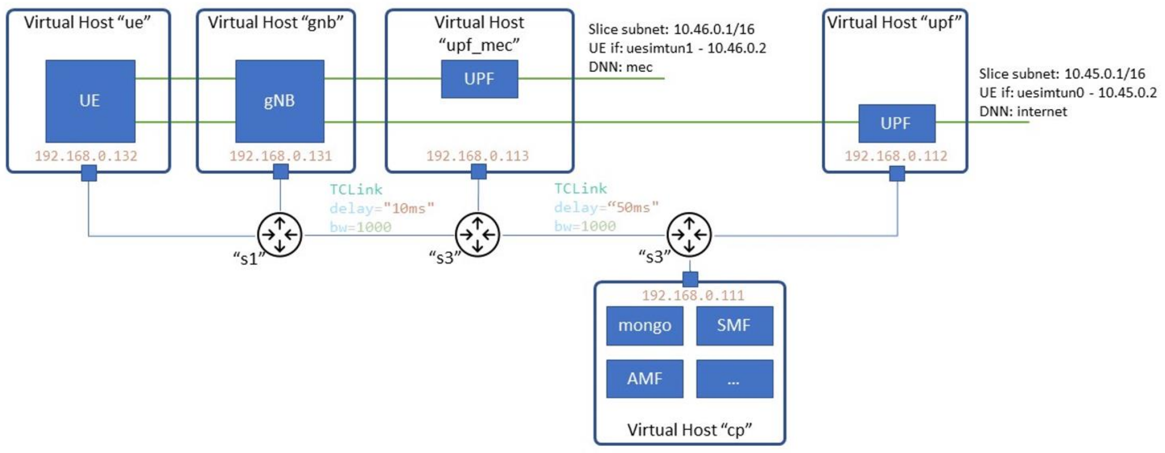


Figure 4: A block diagram on the deployment of 5G in Comnetsemu

Digital Twin-based Prediction module

This module is aimed at predicting relevant scenarios in order to signal potential treats or other performance degradations to the HORSE architecture. In the first implementation, it will be able to detect traffic peaks and potential congestion as well as some types of security attacks.

I/O Interface module

The Digital Twin offers a REST API for interaction with the other modules of the HORSE architecture, as well as for most of the interactions among its internal modules. A Swagger interface is provided to enable fast and efficient testing of the proper operation of all offered functionalities.

Interfaces with other HORSE modules

The Prediction and Prevention Network Digital Twin interacts runtime with two modules of the HORSE architecture: the Early Modeling module and the Distributed Trustable AI Engine module. In both cases the interaction is performed through REST API interfaces and JSON files.

The Early Modeling module provides in input to the NDT information about potential attacks, and it can focus the analysis of the NDT towards specific nodes, by providing IP addresses and ports.

The NDT provides in output to the Distributed Trustable AI Engine any notification about potential anomalies and attacks, in order to trigger a most precise identification of the attack parameters through AI/ML and to select the most appropriate mitigation actions.

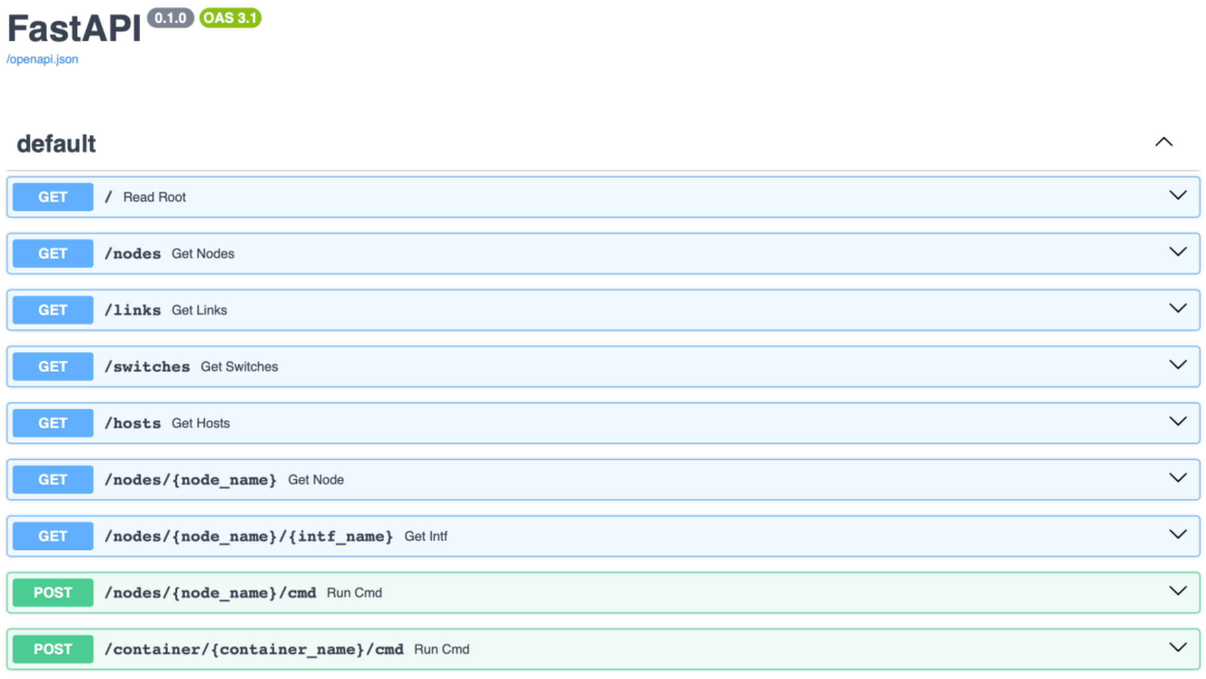
The exposed ports are configurable through a proper "config.ini" file in the software distribution.

Digital Twin Management Interface

The Prediction and Prevention Network Digital Twin offers an internal management interface in order to enable other modules as well as the network manager to interact with the NDT and to trigger actions or perform measurements.

The management interface provides a set of REST APIs, and it is accessible through two TCP/IP ports:

- Port 8501 (default): it enables to control the parameters of the network emulator embedded in the NDT (comnetsemu) and to issue simple commands to collect the status of nodes and containers, and to run commands in the nodes or switches of the topology. This REST interface is developed using Python FastAPI framework, and it is based on SWAGGER. This allows to provide full documentation directly online. As an example:
  - http://192.168.130.9:8501/ provides access to the main internal dashboard (built on Streamlit Python software)
  - http://192.168.130.9:8000/ checks the status of the NDT (if everything is correct, then the answer is a simple JSON file: {"Digital Twin":"Ready"})
  - http://192.168.130.9:8000/docs provides access to the SWAGGER FastAPI interface and helps quick interaction with the NDT (see figure below)
  - http://192.168.130.9:8008/ provides access to the sFlow network monitoring software (for topology and bandwidth analysis)


FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET / Read Root
GET /nodes Get Nodes
GET /links Get Links
GET /switches Get Switches
GET /hosts Get Hosts
GET /nodes/{node_name} Get Node
GET /nodes/{node_name}/{intf_name} Get Intf
POST /nodes/{node_name}/cmd Run Cmd
POST /container/{container_name}/cmd Run Cmd

Figure 5: The SWAGGER REST API of the Prediction and Prevention Network Digital Twin.

- Port 3000 (default): this port is connected to an internal Grafana data rendering instance, which is connected to several container metrics and other performance parameters. Through Grafana it is possible to perform analysis of the data e.g. related to CPU utilization and load for each container in the NDT, or measure throughput on different virtual interfaces.
  The interface is available connecting to the IP address of the P&P NDT on port 3000.

The exposed ports are configurable through a proper "config.ini" file in the software distribution.

Additional details on the execution and configuration of the Prediction and Prevention Network Digital Twin are available in the corresponding section of the github repository of the HORSE project.

3 Development of the Early Modeling framework

The Early Modelling module is developed to supply the Sandboxing module with all the essential data it needs to operate effectively. This module is structured around two core elements: Taxonomy and Attributes. The Taxonomy component focuses on identifying and categorizing potential threats and attack scenarios relevant to the 6G environment. Meanwhile, the Attributes component outlines the methodology and criteria used to assess how these attacks could affect 6G components, as well as to evaluate the effectiveness and consequences of various mitigation and prevention strategies. The overall architecture of the Early Modelling module is illustrated in Figure 6.

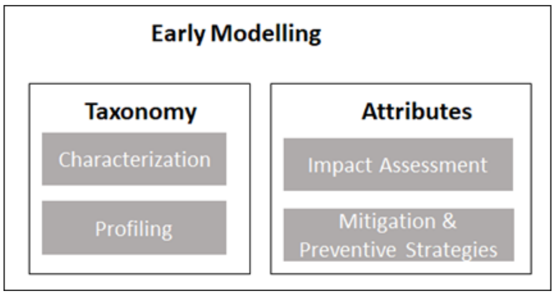


Figure 6: Early Modelling components.

During the initial phase of the project, the Early Modelling module focused on developing a threat model to characterize the different types of 6G cyberattacks. This model was structured around several key components:

- **Vulnerability**: Refers to weaknesses or flaws in a system that can be exploited by adversaries. The meta-model leverages the concept of an *attack surface* to describe these vulnerabilities in detail. This includes aspects such as user equipment, network infrastructure, and exposed services that could be targeted.
- **Organizational Assets**: Encompasses the critical assets and devices within an organization that are of interest to threat actors. These represent the potential targets an adversary seeks to exploit.
- **Threat Actor**: Defined as a malicious entity with the intent and motivation to compromise a system. The meta-model classifies threat actors based on their origin (internal or external to the system) and assesses their skill level using a Likert scale.
- **Tactics, Techniques, and Procedures (TTPs)**: Represents the methods and strategies employed by attackers to achieve their objectives. This includes the overall approach (tactics), specific means of execution (techniques), and concrete steps (procedures) used during an attack.
- **Threat**: A threat is a potential harmful event that arises due to the presence of a vulnerability. It may arise from weaknesses identified in the ENISA threat landscape, specific use case activities, adversarial behaviour, or observable patterns.
- **Cyberattack**: Describes the actual malicious action carried out by a threat actor who exploits system vulnerabilities to cause harm or disruption.
- **Control Actions**: Refers to the set of countermeasures designed to address an attack. These include mitigations, preventive and corrective measures aimed at reducing the risk or impact of an attack.

Deliverable D3.1 [15] provided a detailed description of each element, as well as its type and scope in the HORSE architecture.

In the second period of the project (IT-2), the complete set of attacks considered in HORSE has been modelled. These includes the DNS amplification, NTP DDoS, and DDoS Downlink attacks. Each of these attacks was characterized using the threat model's core elements: threat actor, cyberattack, organization assets, tactics-techniques-procedures (TTP), impact, and information on mitigation and control actions, including the mitigation and preventive actions to be enforced when an attack is detected or predicted.

Figure 7 presents the updated threat model, which incorporates new design requirements to better support both the description of the attacks and the corresponding impact analysis. Specifically, the cyberattack vector element has been extended to support specific details about the attack, as well as the organizational assets to support network-specific features, including details about nodes and ports as defined in the network topology.

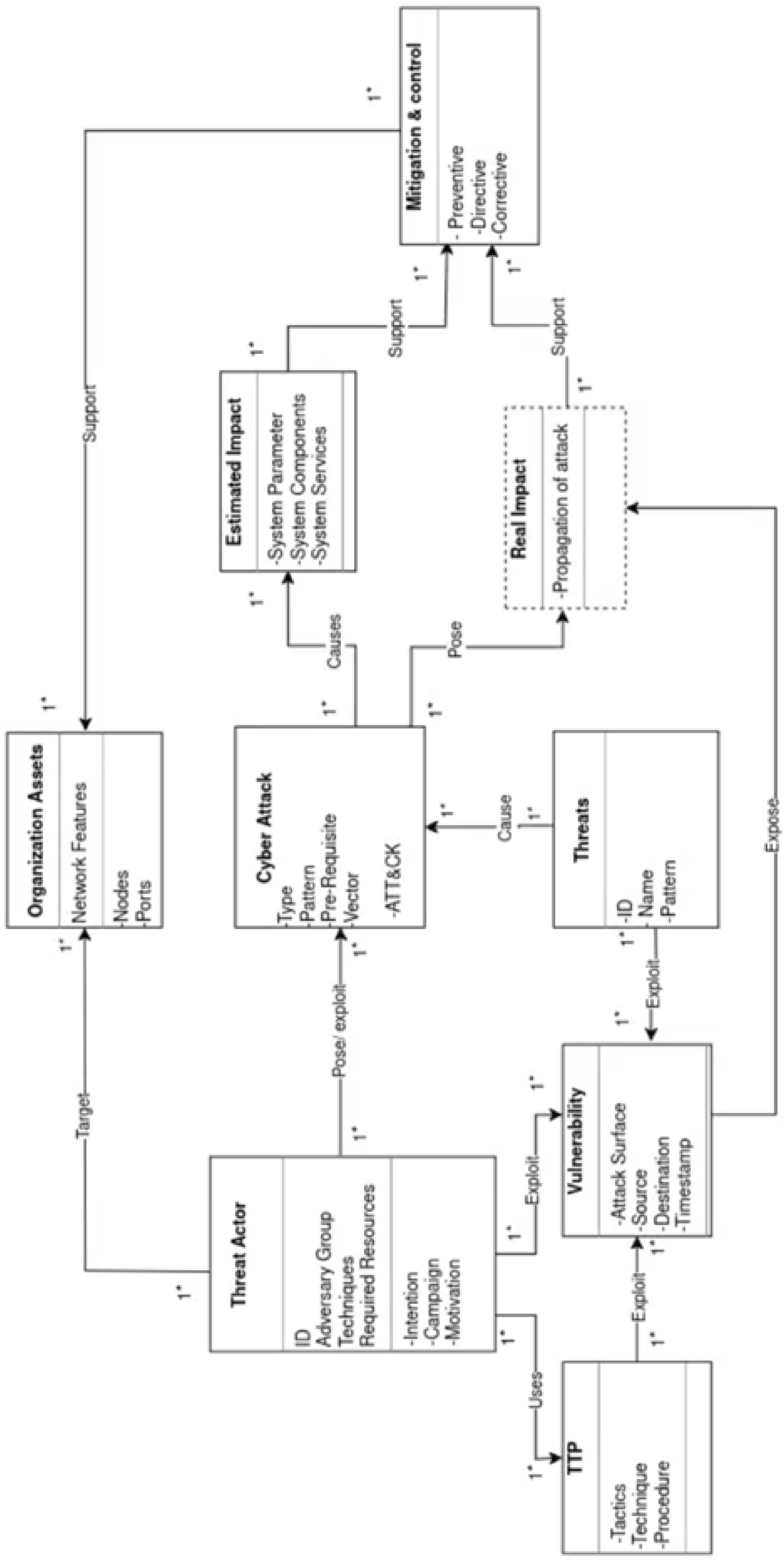


Figure 7: Threat model

3.1 Threat model XML schema

An XML schema has been specified for the meta-model to allow its complete representation. This schema is based on the initial one elaborated during the first phase of the project and it has been updated to support the extensions for the cyberattack vector and organizational asset, as depicted in Figure 8 and Figure 9.

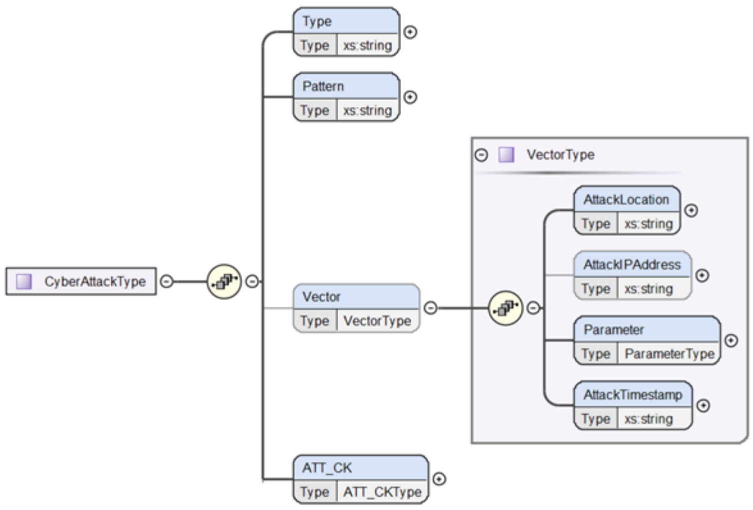


Figure 8: CyberAttackType XML element

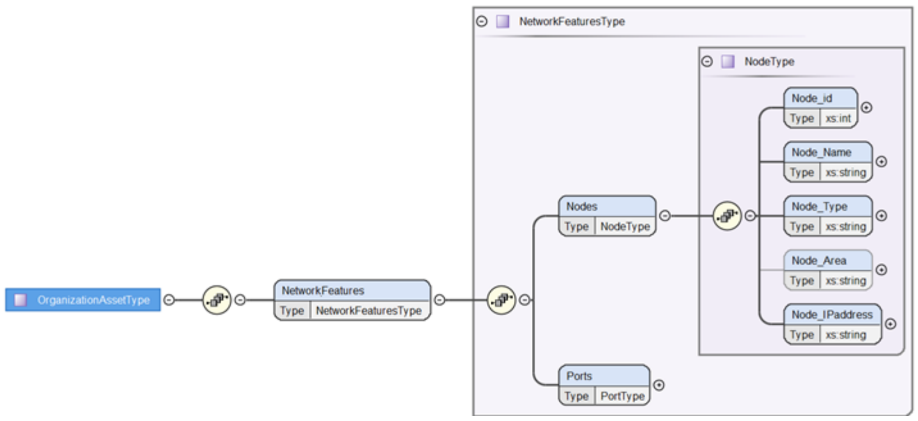


Figure 9: NetworkFeaturesType XML element

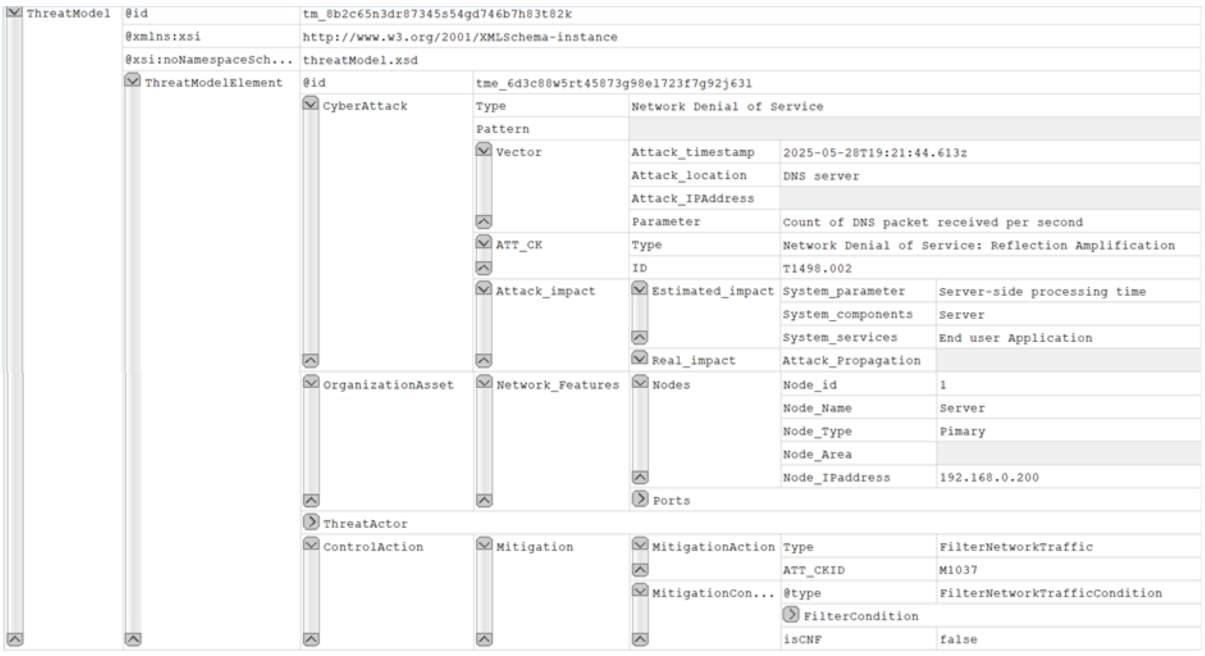
3.2 Attacks modelling in the HORSE framework

This section presents the modelling done for the attacks including the DNS amplification and NTP DDoS attacks.

3.2.1 DDoS DNS amplification

This section presents the modelling of the DNS amplification attack to demonstrate how the threat model captures this attack scenario. In the proposed XML schema, the

`<ThreatModel>` element encapsulates all relevant components depicted in Figure 7, while also integrating additional information necessary for Prediction and Prevention DT to monitor key parameters during the prediction of the attack. These parameters are specified under the `<vector>` element, as illustrated in Figure 8. The `<CyberAttack>` element serves as a bridge to map the attack scenario to known adversarial tactics, techniques, and procedures (TTPs) by referencing established threat intelligence frameworks. Specifically, to align with the MITRE ATT&CK framework, we associate the attack with its corresponding technique by including the relevant `ID` and `type` attributes. For the DNS amplification scenario, this is represented as `ID="T1498.002"` and `type="Network Denial of Service: Reflection Amplification"`.

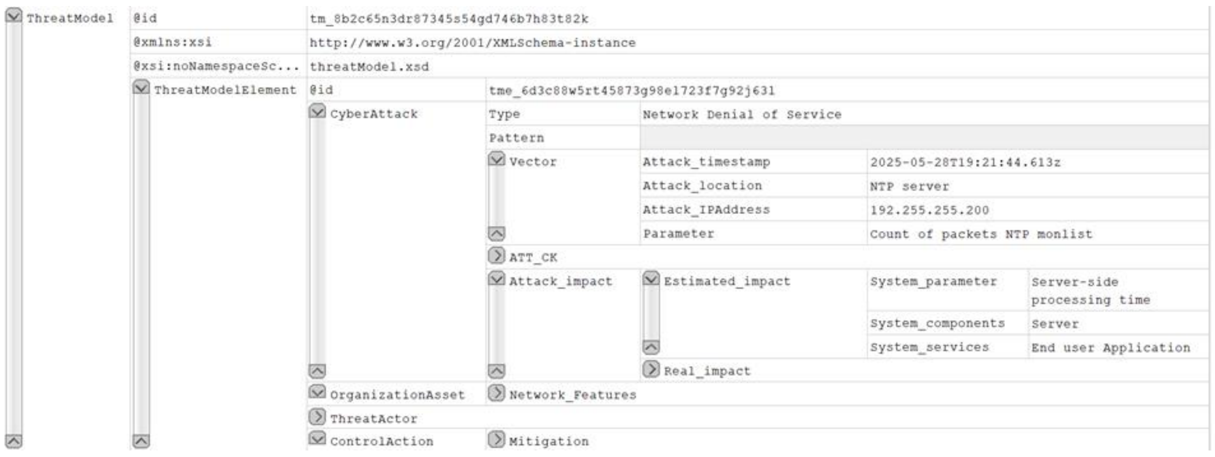


```
ThreatModel  @id                         tm_8b2c65n3dr87345s54gd746b7h83t82k
             @xmlns:xsi                  http://www.w3.org/2001/XMLSchema-instance
             @xsi:noNamespaceSch...      threatModel.xsd
             ThreatModelElement  @id     tme_6d3c88w5rt45873g98e1723f7g92j631
                   CyberAttack   Type    Network Denial of Service
                                 Pattern
                                 Vector          Attack_timestamp    2025-05-28T19:21:44.613z
                                                 Attack_location     DNS server
                                                 Attack_IPAddress
                                                 Parameter           Count of DNS packet received per second
                                 ATT_CK          Type                Network Denial of Service: Reflection Amplification
                                                 ID                  T1498.002
                                 Attack_impact   Estimated_impact  System_parameter   Server-side processing time
                                                                   System_components  Server
                                                                   System_services    End user Application
                                                 Real_impact       Attack_Propagation
                   OrganizationAsset  Network_Features  Nodes      Node_id            1
                                                                   Node_Name          Server
                                                                   Node_Type          Pimary
                                                                   Node_Area
                                                                   Node_IPaddress     192.168.0.200
                                                        Ports
                   ThreatActor
                   ControlAction      Mitigation  MitigationAction   Type       FilterNetworkTraffic
                                                                     ATT_CKID   M1037
                                                  MitigationCon...   @type      FilterNetworkTrafficCondition
                                                                     FilterCondition
                                                                     isCNF      false
```

Figure 10: DDoS DNS amplification attack

3.2.2 DDoS NTP

The proposed threat model also has been used to model the DDoS NTP attack, as illustrated in Figure 11. In this scenario, the attacker exploits a Network Time Protocol (NTP) server, typically by leveraging the monlist command to amplify traffic. The DDoS NTP XML file suggests to the Prediction and Prevention DT to monitor the volume of packets associated with the NTP monlist, as illustrated in Figure 11. To prevent the system against such an attack, it is necessary to enforce mitigation strategies and implement conditions to filter UDP-based traffic.



```
ThreatModel  @id                     tm_8b2c65n3dr87345s54gd746b7h83t82k
             @xmlns:xsi              http://www.w3.org/2001/XMLSchema-instance
             @xsi:noNamespaceSc...   threatModel.xsd
             ThreatModelElement      @id                 tme_6d3c88w5rt45873g98e1723f7g92j631
                                     CyberAttack         Type              Network Denial of Service
                                                         Pattern
                                                         Vector            Attack_timestamp    2025-05-28T19:21:44.613z
                                                                           Attack_location     NTP server
                                                                           Attack_IPAddress    192.255.255.200
                                                                           Parameter           Count of packets NTP monlist
                                                         ATT_CK
                                                         Attack_impact     Estimated_impact    System_parameter    Server-side
                                                                                                                   processing time
                                                                                               System_components   Server
                                                                                               System_services     End user Application
                                                                           Real_impact
                                     OrganizationAsset   Network_Features
                                     ThreatActor
                                     ControlAction       Mitigation
```

Figure 11: DDoS NTP attack

3.3 I/O interface for the Early Modeling module

The early modelling framework offers a FastAPI endpoint for the other modules of the HORSE architecture. The JSON input and XML output have been defined for the API endpoints. The exposed ports will be configurable through a proper “config.json” file in the software distribution. Moreover, the early modelling is configured to connect with the default network generated by the Knowledge base and communicate with the API to fetch the mitigation actions. The url and port can be specified in the config.json file. The Docker compose file configuration might need to be modified as per the configuration of the Knowledge base.

Input API endpoint can be triggered using the Curl command as presented below.

```
curl --location 'http://127.0.0.1:8000/xml-scheme/'
--header 'Content-Type: application/json'
--data '{ "CyberAttack": { "Type": "DDoS Downlink", "Vector": {
"attack_timestamp": "2024-04-28T19:21:44.613Z", "attack_location": "DNS
server", "Asset_IPAddress": "192.255.255.200", "Parameters": {
"Description":"11 of DNS packet received per second", "Protocol":"12",
"Flag":"123", "Duration":"321" } } }, "OrganizationAssets": {
"Network_feature": { "Node": { "Node_Id": "1", "Node_name": "Server",
"Node_Type": "primary", "Node_Area": "", "Node_IPAddress": "192.168.0.200"
}, "Ports": { "Port_Id": "1", "Port_Number": "53", "Port_Type": "UDP",
"Port_Status": "Open" } } },
 "ThreatActor": { "Source":"", "ThreatActor_IPAddress": "",
"ThreatActor_Group": "", "ThreatActor_Technique": "",
"ThreatActor_Intension": "" }, "TTP": { "Tactics": "", "Techniques": "",
"Procedure": "" }, "Vulnerability": { "Source": "", "Destination": "",
"Timestamp": ""

}}'
```

The XML file will be sent to the specified endpoint whenever it is generated. The endpoint information needs to be added in the config.json file.

```
curl --location --request POST 'http://192.168.130.51:8000/fetch-xml-schema'
--header 'Content-Type: application/json'
--data ''
```

4 Development of the Distributed Trustable AI Engine

4.1 Introduction

The Distributed Trustable AI Engine (DTE) is a main component of the HORSE architecture that sends the appropriate intents to the IBI after an attack is identified. DTE receives inputs from the DEME at periodic time intervals, with a predefined confidence level as well the identified type of the attack and the involved nodes of the HORSE platform. This information is internally processed in the DTE via AI/ML approaches in order to identify the appropriate intents per case. The advices from the DEME are sent using REST HTTP requests to the DTE. In the same context, DTE can also receive data in the form of policies from PAG, via REST APIs.

DTE performs data management prior to the actual training, by employing the appropriate policies for anomaly detection (tampered data), as well as data anonymisation. In addition, DTE guarantees compliance of the proposed solutions with the policies module. In the next step, mitigation measures and methodologies from well-established knowledge bases, such as the MITRE ATT&CK are exploited from both the DEME and DTE in order to build the appropriate mitigation intents.

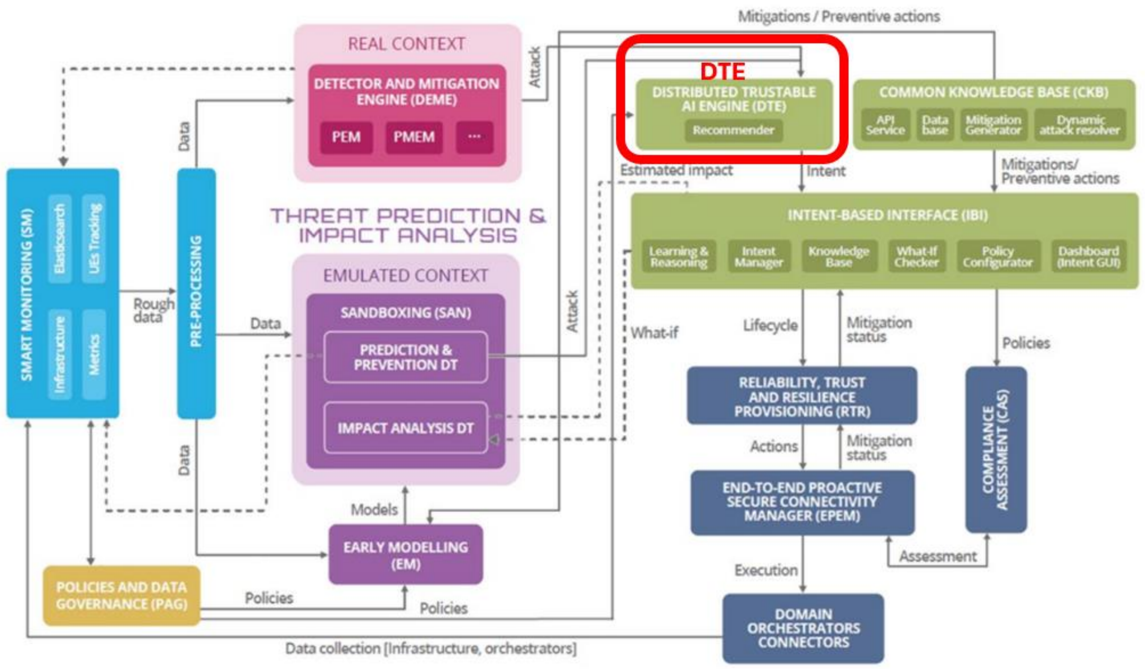


Figure 12: The DTE component within the HORSE architecture

4.2 Internal components of the DTE

The internal components of the DTE are shown in Figure 13. These include the NWDAF aggregator, the Data Processing Module, the ML Model Training Module, the ML Model Evaluator, the ML Model Repository as well as the Intent Creator. The NWDAF aggregator receives all data from distinct NWDAF instances that are deployed within the HORSE network. In Data Processing Module, since the collected data might be heterogeneous, they are properly processed in order to be used for the actual ML model training. In this module, the training of various ML approaches can be supported. As it will be also explained in the next section, performance metrics include F1-score, accuracy, etc which are evaluated via the ML Model Evaluator. The best performing models are stored in the ML model repository. At the final stage, the intent to be send in the IBI is created via the Intent Creator module.

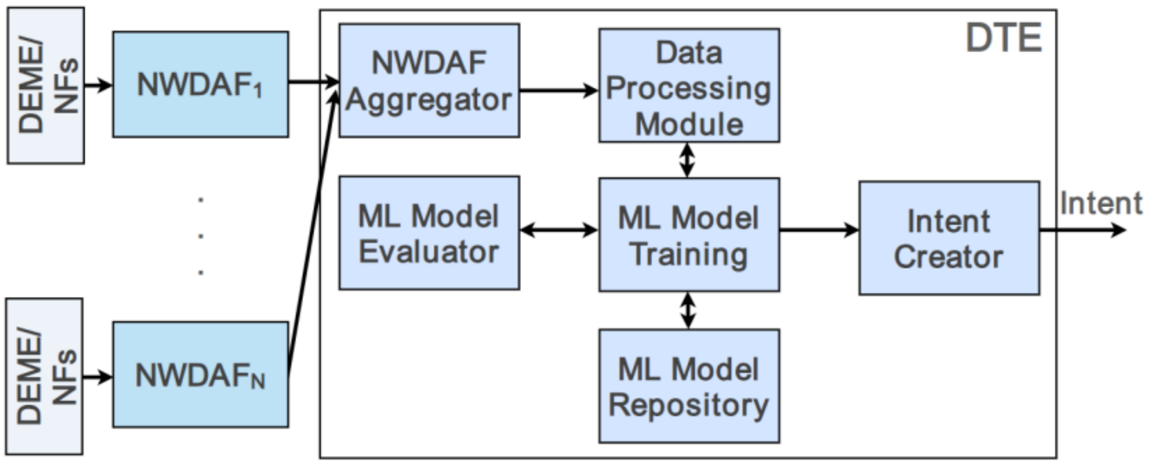


Figure 13 : The internal components of the DTE.

4.3 Datasets and performance evaluation of the DTE

The ML training is the main module of the DTE where various models are trained for different types of attacks. These will include supervised, unsupervised, and deep reinforcement learning approaches [16]. For this purpose, various datasets were exploited, representing diverse attacks and network topologies. These datasets are provided i) from HORSE partners, ii) from the NKUA Open5GS and UERANSIM-based testbed, being able to replicate 5G core network attacks, as well as attacks on the 5G RAN, and iii) open datasets that have been used from relevant works on 5G attack scenarios.

At this final stage, four different datasets from the literature have been analysed together with ML model training and evaluation:

- The first one is a synthetic 5G cellular network data for NWDAF [17], that is based on Open5GS and UERANSIM. In this context, a topology with a fixed number of subscribers and cells with different traffic patterns and anomalies has been considered, where the anomaly is defined as an unexpectedly high network traffic compared to the average network traffic, fading and stabilizing in time.
- The second dataset is the 5GAD-2022 5G attack detection dataset [18], that is based on Free5GC and UERANSIM. In this case, two types of intercepted network packets are included: "normal" network traffic packets and "attack" packets from attacks against a 5G Core implemented with free5GC. The captures were collected using Tshark or Wireshark on 4 network interfaces (N2, N3, N4, N6) (AMF, gNB, UPF, SMF, DN) within the 5G core. 10 attacks were implemented, mainly relying on REST API calls to different parts of the core.
- The third dataset [19] was generated on an Open5GS and UERANSIM-based testbed. Here, an SMF instance networked in parallel to the original network function acts as the attacker’s entry point to the virtualised infrastructure and targets the N4 interface between

the SMF and the UPF. The hijacked SMF executes the cyberattacks against the UPF. In order to obtain this data set, the network traffic data of each entity/device was captured through Tshark for each network function and radio element.

- The fourth data set, titled 5G Network Intrusion Detection Dataset (NIDD) [20] contains data in both packet-based format as well as in flow-based formats. 5G-NIDD is generated using the 5G Test Network (5GTN) in Oulu, Finland, thus providing a close resemblance to a real network scenario. 5G-NIDD presents a combination of attack traffic and benign traffic under different attack scenarios, falling into the Distributed Denial of Service (DDoS) and Port Scan/Reconnaissance categories.

For these datasets, various ML models have been evaluated by the ML model evaluator module for predefined ML metrics, such as accuracy and F1-score. It should be noted that apart from evaluating the anomaly detection performance of different ML models, e.g. support vector machine (SVM) with binary kernel for the N4 interface attacks, these attacks have already been replicated at the NKUA testbed and data are collected, as well as logs from various NFs, i.e. AMF, SMF and UPF. After the initial ML model evaluation phase is over, two distinct actions can take place: i) retraining of the ML model in case its performance is below the desired level, or ii) storage of the model in the ML repository, for retrieval in future potential attacks.

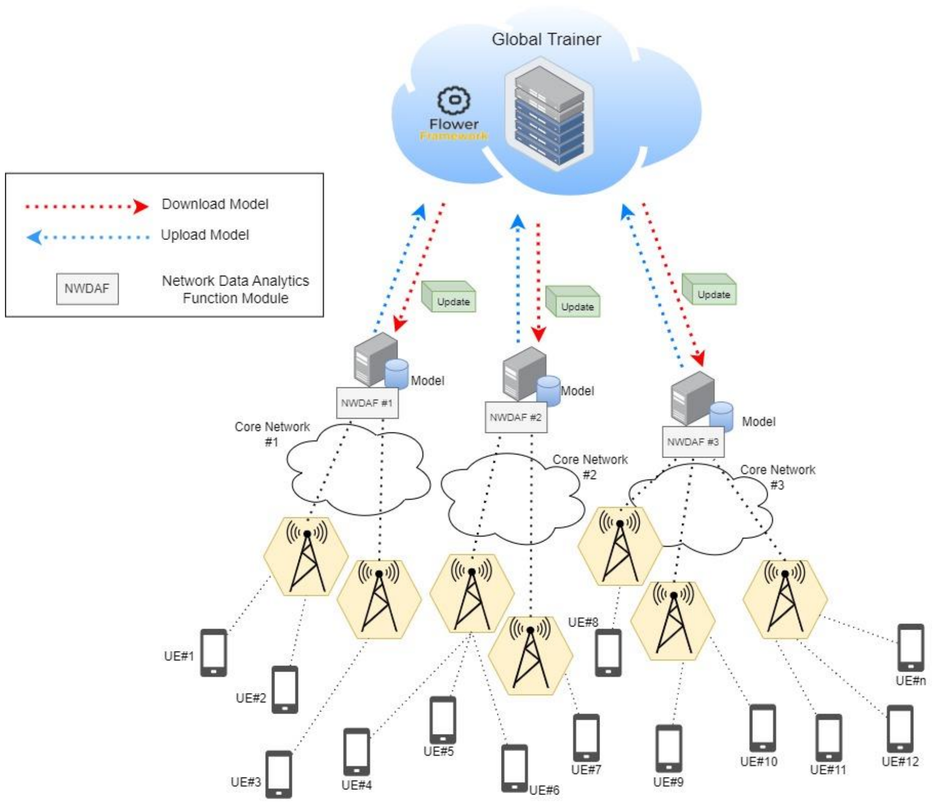


Figure 14: The FLOWER concept in DTE.

4.4 Final extensions with respect to IT-1

In the final release of the DTE, federated learning (FL) was also applied, where each NWDAF instance is responsible for data collection and aggregation in a distinct set of mobile nodes, as shown in Figure 14 In this case, there are multiple DTE instances per subgroup of nodes, where each one trains locally the corresponding models with the available datasets. Afterwards, the master DTE model with the NWDAF aggregator is responsible for updating the global parameters and informing the individual nodes for their updated values. For this purpose, the FLOWER concept was applied, that can train multiple nodes in an FL fashion [21]. In the Table below, all major differences among IT-1 and IT-2 are summarized.

Feature	**Original: HORSE + BentoML**	**Updated: HORSE + BentoML + FLower (Federated)**
Architecture Type	Centralized	Federated Learning (1 server, 2 clients)
Data Locality	Central — data aggregated at one node	Decentralized — data stays on each simulator
Model Training	Trained offline, served via BentoML	Federated training with model updates per client
Model Serving	BentoML API server (REST or gRPC)	BentoML inside each Flower client for inference
Model Update Flow	Manual retraining / offline CI/CD	Auto via FLower round-based aggregation
Privacy	Low — raw data required at central point	High — raw data stays local
Network Dependency	Low (no sync required between machines)	High (needs synchronization with Flower server)
Real-Time Detection	Possible with BentoML service endpoint	Possible per client; global insight delayed
Deployment	Dockerized BentoML service	Dockerized Flower client + BentoML per node

Aspect	Before (Centralized: HORSE + BentoML)	Now (Federated: HORSE + BentoML + FLoWeR)
1. Data Handling	- All data collected centrally - Preprocessed in one place - HORSE logic applied globally	- Data remains local per simulation - Each node runs HORSE independently - Only model weights are shared (privacy-preserving)
2. Model Serving	- bentoml.Model exposed via REST/gRPC - Centralized API for external inference	- Each Flower client runs BentoML locally - Inference is local-only unless explicitly exposed
3. Model Training	- Manual or CI/CD-triggered training - Static or offline updates - HORSE rules embedded into periodic model refresh	- Training distributed via federated rounds - Each client trains on local recent data - Server aggregates updates and redistributes
4. Security/Resilience	- Central model is a single point of failure - Model poisoning or threshold flaws affect the whole system	- Semi-autonomous clients - Failures or poisoned updates are localized - More resilient to region-specific anomalies or attack vectors
5. Deployment	- Single Docker container per deployment - Simple stack (HORSE + model + BentoML) - Easy with Docker Compose or K8s	- Multiple coordinated containers: 1 Flower Server + N Flower Clients - Each client runs its own BentoML & model stack - Requires Docker networking across instances
6. Monitoring /Eval	- Centralized logging and metrics collection - Easy benchmarking, debugging, versioning	- Decentralized monitoring per Flower client - Flower server only sees updates, not full context - Local metrics must be pulled manually or exposed through logging layers

5 Development of the Policies and Data Governance

5.1 Overview

The Policies and Data Governance (PAG) component is a data processing service built with NestJS[26]. It is designed to harvest Packet Capture (PCAP) data in JSON format from the Smart Monitoring component. Subsequently, the PAG handles and anonymises sensitive information, and finally it stores the anonymised PCAP data back into the storage indexes of the Smart Monitoring component. The service uses Redis[27] and Bull[28] for job queue management and scheduling, ensuring efficient and scalable processing.

5.2 Key Features

PCAP Data in JSON Format

The input to fetch the service is PCAP data which is converted into JSON format. Each record contains network-related metadata, such as “from” and “to” IP addresses, payload, timestamps, and protocol details.

Service Activation via Cron Jobs

A cron job is scheduled to run periodically. The cron job enqueues tasks into a Redis-backed queue, which are then processed asynchronously.

Ad-hoc Service Activation

The component has implemented a manual trigger for the service, in order to enable ad-hoc use and testing.

Data Read/Write Operations

Read: The service fetches PCAP data from the storage of the Smart Monitoring component, which is an Elasticsearch index.

Write: The service writes anonymised PCAP data back to a separate Elasticsearch index of the Smart Monitoring component.

Custom Anonymisation

Sensitive fields such as IP addresses, payload content and International Mobile Equipment Identity (IMEI) numbers are anonymised using custom logic. The anonymisation logic is modular and reusable, ensuring flexibility for different data structures.

5.3 Technologies Used

Framework: The application is built using NestJS, a progressive Node.js framework for building scalable server-side applications.

Modules: The application is modularised.

- ElasticModule: handles all the interactions with Elasticsearch.
- Provides methods for:
  - Harvesting PCAP data using the scroll API.

  - Writing anonymised PCAP data back to the Elasticsearch index.

- QueueJobsModule: manages job queueing and processing.
- Functions:
  - JobSchedulerService: Periodically enqueues jobs using a cron job.
  - QueueJobsProcessor: Processes jobs from the queue.

- AnonymisationModule: contains the logic for anonymising sensitive fields in the PCAP data.
- Modular and reusable for different data structures.

Redis: It acts as a backend for job queue management. Redis is integrated with the Bull library for managing and processing jobs.

Bull: This is a queueing library for handling job scheduling and processing. Jobs are enqueued by the cron job and processed by a queue processor.

Environment Variables: the service uses the following environment variables, typically stored in an .env file.

```
# Redis Configuration
REDIS_HOST=redis
REDIS_PORT=6379

# Elasticsearch Configuration
ELASTICSEARCH_URL=http://elasticsearch:9200
ELASTICSEARCH_INDEX=pcap-data-local
ELASTICSEARCH_ANONYMIZATION_INDEX=pcap-data-anonymization
```

5.4 How It Works

Data Flow

- *Input*: The service harvests PCAP data in JSON format from the Elasticsearch index of the Smart Monitoring component.
- *Processing*:
  - The PCAP data is fetched in batches using Elasticsearch scroll API.
  - Sensitive fields (e.g., “from” and “to” IP addresses, payload content, IMEI) are anonymised using custom logic.
- *Output*: The anonymised data is written to a separate Elastisearch index of the Smart Monitoring component.

Communication

A cron job is scheduled to run periodically (configurable; currently set to run every day at midnight). The cron job enqueues tasks into jobs using the Bull package. The queue processor fetches the tasks and processes them asynchronously.

Custom Anonymisation Logic

IP Addresses: masked with X (e.g. **X.X.X.X**)

Payload: masked with a string **[REDACTED_PAYLOAD]**

IMEI: masked with a string **[ANONYMISED_IMEI]**

Sequence Diagram

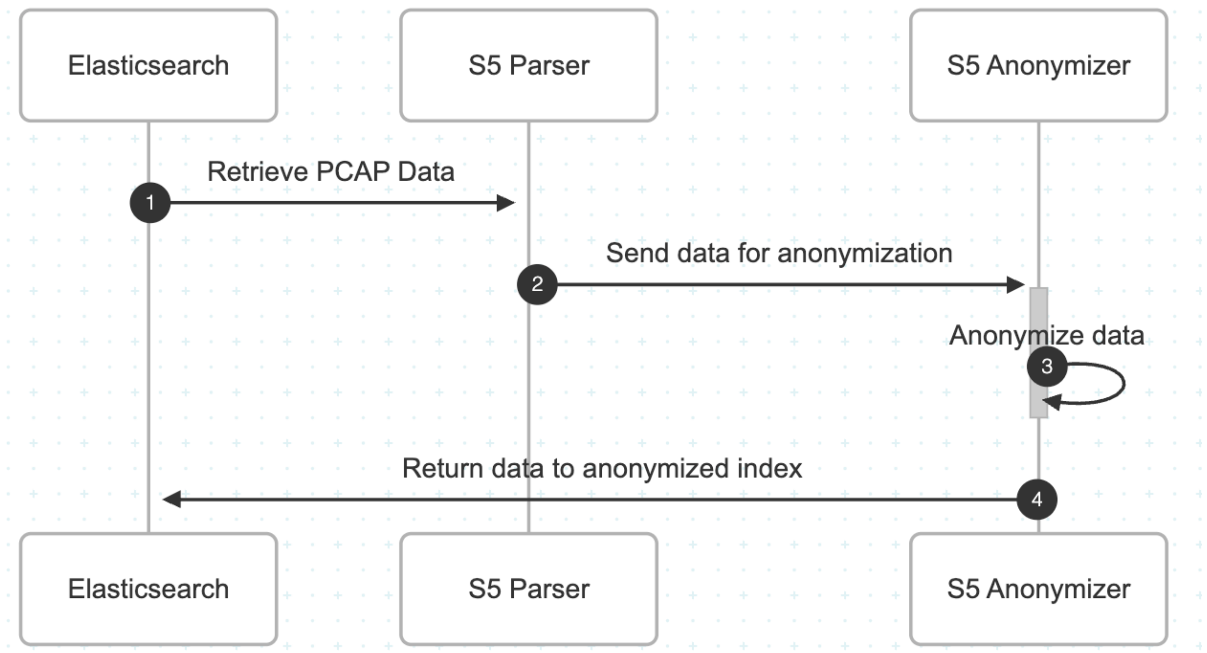


Figure 15: Sequence Diagram

6 Development of the Threat Detector and Mitigation Engine

6.1 Threat Detector and Mitigation Engine Features

The functionalities of the Threat Detector and Mitigation Engine (DEME) module are fully consistent with the specifications outlined in the IT-1 deliverable (D3.1 HORSE Platform Intelligence developed (IT-1) [15]) and remain aligned with the architectural requirements and design principles established in the WP2 deliverables, ranging from D2.1 [6] (HORSE Landscape: Technologies, state of the art, AI policies and requirements (IT-1)) and D2.2 [7] (HORSE Architectural Design (IT-1) through to the most recent D2.3 [8] (HORSE Landscape: Technologies, state of the art, AI policies and requirements (IT-2) and D2.4 [9] (HORSE Landscape and Architectural Design).

In the context of the HORSE architecture, and as illustrated in the overall block diagram (recalled here for clarity in Figure 16), the system is organized into three key conceptual layers:

- Intent-Based Interface (IBI) – This layer serves as the entry point for high-level directives, allowing network administrators or intelligent software agents to express desired outcomes without needing to manage low-level configuration details. Its primary role is to abstract and simplify network control through intent-driven interactions.

- AI Secure and Trustable Orchestration (STO) – Operating as the intermediary control logic, this layer is responsible for the dependable orchestration of network resources. It ensures that the intents expressed via the IBI are correctly interpreted and translated into executable actions, enforcing the corresponding policies while maintaining system reliability and trustworthiness.

- Platform Intelligence Layer (PIL) – This layer introduces advanced intelligence and autonomy into the system. It includes multiple sub-modules capable of analyzing, predicting, and optimizing network behavior. Among these, the DEME module (situated in the “Real Context” section) is specifically designed to identify, assess, and respond to security threats in real time, leveraging network behavior analysis and anomaly detection.

The DEME module plays a critical role in this architecture by enabling the platform not only to detect threats but also to initiate appropriate mitigation strategies autonomously, thereby enhancing the overall security and resilience of the HORSE system.

Co-funded by the European Union

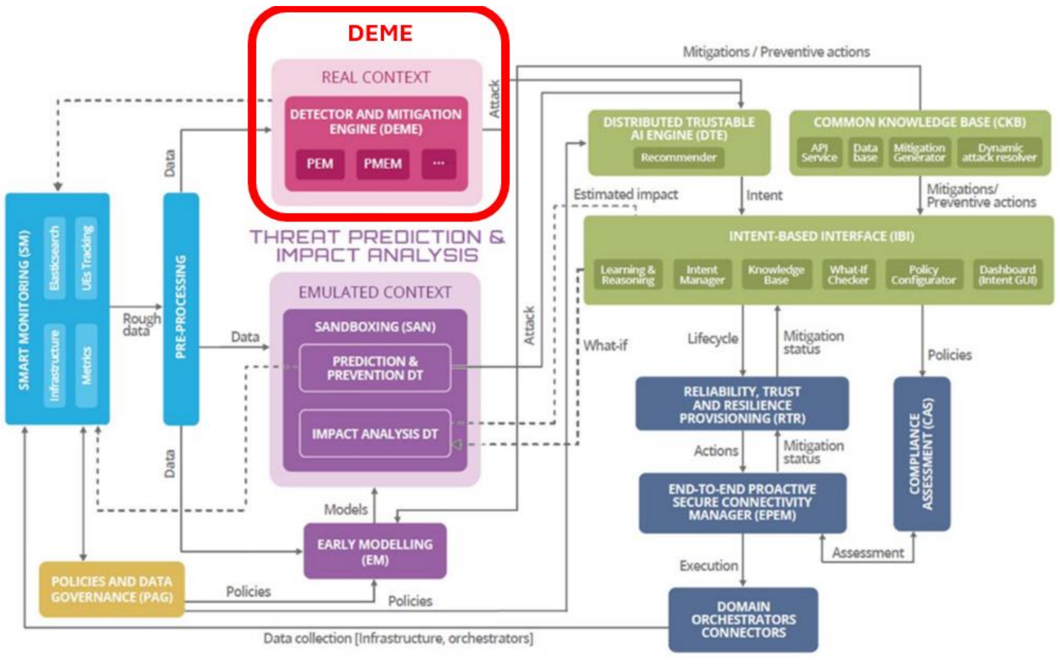

DEME
REAL CONTEXT
DETECTOR AND MITIGATION ENGINE (DEME)
PEM
PMEM
...
Attack
THREAT PREDICTION & IMPACT ANALYSIS
EMULATED CONTEXT
SANDBOXING (SAN)
PREDICTION & PREVENTION DT
IMPACT ANALYSIS DT
SMART MONITORING (SM)
Elasticsearch
UEs Tracking
Infrastructure
Metrics
PRE-PROCESSING
Rough data
Data
Models
EARLY MODELLING (EM)
POLICIES AND DATA GOVERNANCE (PAG)
Policies
Data collection [Infrastructure, orchestrators]
Mitigations / Preventive actions
DISTRIBUTED TRUSTABLE AI ENGINE (DTE)
Recommender
COMMON KNOWLEDGE BASE (CKB)
API Service
Data base
Mitigation Generator
Dynamic attack resolver
Estimated impact
Intent
Mitigations/ Preventive actions
INTENT-BASED INTERFACE (IBI)
Learning & Reasoning
Intent Manager
Knowledge Base
What-if Checker
Policy Configurator
Dashboard (Intent GUI)
What-if
Lifecycle
Mitigation status
Policies
RELIABILITY, TRUST AND RESILIENCE PROVISIONING (RTR)
Actions
END-TO-END PROACTIVE SECURE CONNECTIVITY MANAGER (EPEM)
COMPLIANCE ASSESSMENT (CAS)
Assessment
Execution
DOMAIN ORCHESTRATORS CONNECTORS

Figure 16: DEME sub-module in the overall architecture [9].

6.2 Threat Detector and Mitigation Engine Interfaces

6.2.1 Ingress Interface

The Ingress interfaces have not been modified since the previous iteration (IT-1) D3.1 [15], as the evolution of the project did not require any changes in this area. As a result, the efforts were focused on testing and validating the existing interfaces

6.2.2 Egress Interface

The same applies to the egress interfaces, for which no modifications or additions were necessary; the efforts focused about testing and validation activities

6.2.3 Additional Interfaces (E.g. Digital Twin)

As for potential additional interfaces between the DEME module and other subsystems within the framework—interfaces that were not defined during the first iteration and were not further explored during IT-2, which focused on other aspects as detailed in the WP2 deliverables [6],[7],[8], and [9]—these could be easily integrated in future evolutions of the HORSE framework beyond the project’s official end. This is made possible by the fact that the DEME module has been developed using modern, modular, and extensible software design principles.

6.3 DEME Implementation overview

6.3.1 Background

This section describes how to check prerequisites, install the toolkit and verify it has been correctly installed. A basic understanding of shell scripting and Docker is assumed.

The following terminology is used throughout:

- Client: the host from which commands are issued to the Server.
- Server: the host where workloads are executed.

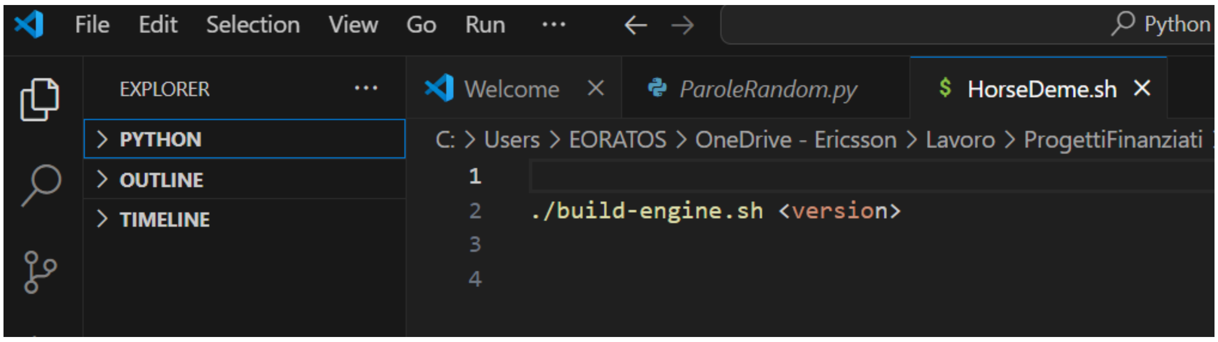
All commands are intended to be run from the Client. It is assumed that Client and Server are separate hosts.

6.3.2 Prerequisites

- Docker must be installed on your local machine.
- Navigate to the root folder of the toolkit in order to proceed with the installation process.

6.3.3 Installation

- Build the Docker image for the server by running the following script:

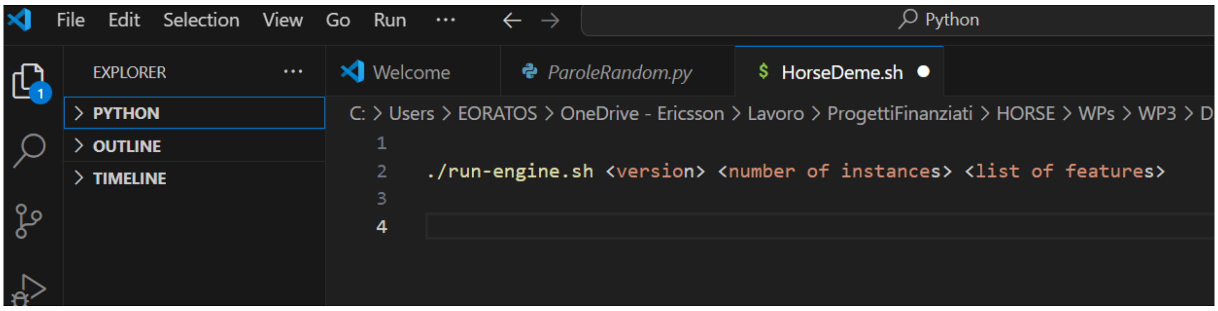


File Edit Selection View Go Run ···
Python
EXPLORER
PYTHON
OUTLINE
TIMELINE
Welcome
ParoleRandom.py
HorseDeme.sh
C: > Users > EORATOS > OneDrive - Ericsson > Lavoro > ProgettiFinanziati

```
./build-engine.sh <version>
```

Figure 17: Docker Image building script

- Start the server using the command:



File Edit Selection View Go Run ···
Python
EXPLORER
PYTHON
OUTLINE
TIMELINE
Welcome
ParoleRandom.py
HorseDeme.sh
C: > Users > EORATOS > OneDrive - Ericsson > Lavoro > ProgettiFinanziati > HORSE > WPs > WP3 > D

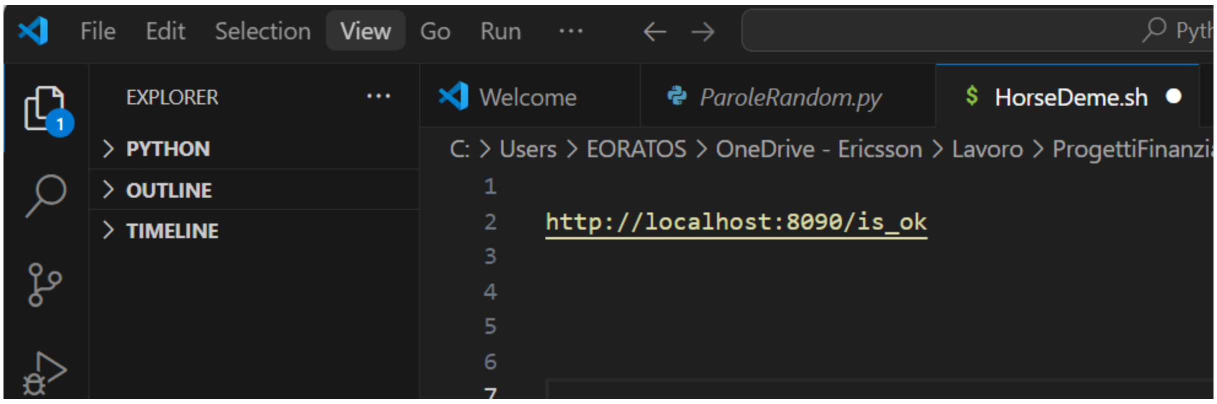
```
./run-engine.sh <version> <number of instances> <list of features>
```

Figure 18: Server start command

Where :

 - <version> is the minor version number.
 - <number of instances> specifies how many nodes are to be monitored,
 - <list of the features to be monitored> refers to the features to be monitored (e.g., NEF or NEF,NTP).

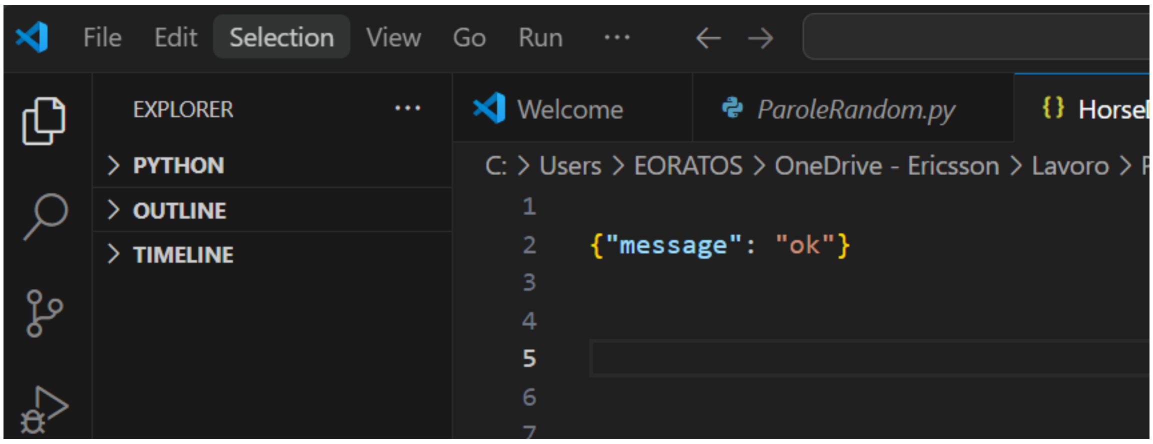
- Verify that the server is running correctly by sending a GET request to:



```
http://localhost:8090/is_ok
```

Figure 19: Correct running state vérification

- A successful response should return:



```
{"message": "ok"}
```

Figure 20: Expected response

6.3.4 Interfaces

1. Management Methods

No changes on respect to deliverable 3.1 [15].

1. Operational Methods

No changes on respect to deliverable 3.1 [15] except for the following API that has been added:

HTTP Method	Path	Action
POST	/reset_db	Reset the training to original values removing the data coming from the runtime elaboration

2. Extraction Methods

No changes on respect to deliverable 3.1 [15].

6.3.5 Usage

No changes on respect to deliverable 3.1 [15].

6.3.6 Debugging tools

No changes on respect to deliverable 3.1 [15].

6.3.7 API-based attacks: unauthorized access to the Network Exposure Function (NEF)

The DEME threat detection module was designed from the outset to incorporate the most advanced Machine Learning technologies, aiming to address the ever-evolving landscape of cyber threats with innovative approaches. Its goal is to detect a broad range of attack vectors, including the increasingly dangerous and unpredictable zero-day attacks. During the second phase of the project, DEME was thoroughly tested against several types of threats to assess its effectiveness. While post-integration testing within the complete platform (covered extensively in WP5, which includes HORSE integration, use case deployment, platform validation, and final release) is absolutely crucial, it is also best practice to thoroughly test each module in isolation during development.

. In Python, this typically involves writing unit tests (e.g., using frameworks like unittest or pytest) and also conducting more advanced integration-level tests that simulate how the module interacts with external systems or services. These represent two increasingly sophisticated levels of validation that allow for comprehensive debugging of the module’s internal logic and its interfaces. By ensuring robustness early on, integration efforts can then focus exclusively on higher-level platform-specific concerns, rather than module-level issues that could have been addressed in isolation.

In line with this approach, DEME has followed all recommended standalone testing practices under WP3, which focuses on the module-level validation phase, prior to integration. One particularly concerning scenario examined—especially relevant for future 6G networks—is the so-called “unauthorized access to the Network Exposure Function (NEF),” as described by the MITRE framework [25]. In such cases, an attacker controlling an external Application Function (AF) may exploit a fraudulent OAuth access token to illegitimately invoke NEF services. NEF is a critical component in mobile networks, exposing sensitive functionalities—such as device analytics, user traffic routing, location tracking, and mobility events—to authorized third-party applications using secure protocols like TLS and OAuth 2.0. However, a malicious AF can bypass these safeguards by presenting a stolen or manipulated token, gaining access to sensitive data and potentially enabling further attacks, such as man-in-the-middle (MITM) or eavesdropping. These types of intrusions are particularly dangerous because they are often discovered only after a long time, causing significant damage in terms of data breaches and privacy violations. As highlighted in specialized literature [24], API-based attacks now represent one of the most serious security threats to modern businesses, as APIs offer direct access to key data and services. Attackers are increasingly aware of the widespread use and known vulnerabilities of these interfaces. While code injection attacks are still common—typically requiring some prior knowledge—brute force attacks remain among the most widely used due to their simplicity and effectiveness. In a brute force attack, an attacker bombards an API or application with repeated requests, attempting to guess authentication credentials, secrets, or access tokens through trial and error. In light of these growing threats, DEME has already addressed and evaluated these new forms of attacks during Iteration 2, ensuring it is robust and ready for integration. The following image illustrates an example snapshot of the specific tests performed.

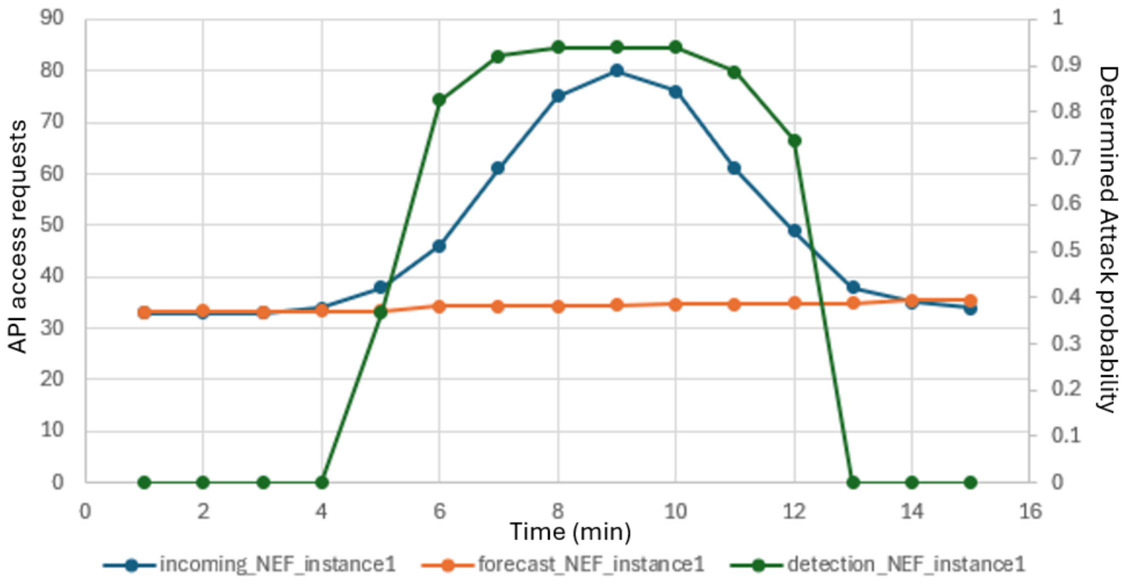


Figure 21: API based attack

6.3.8 Exploratory activity

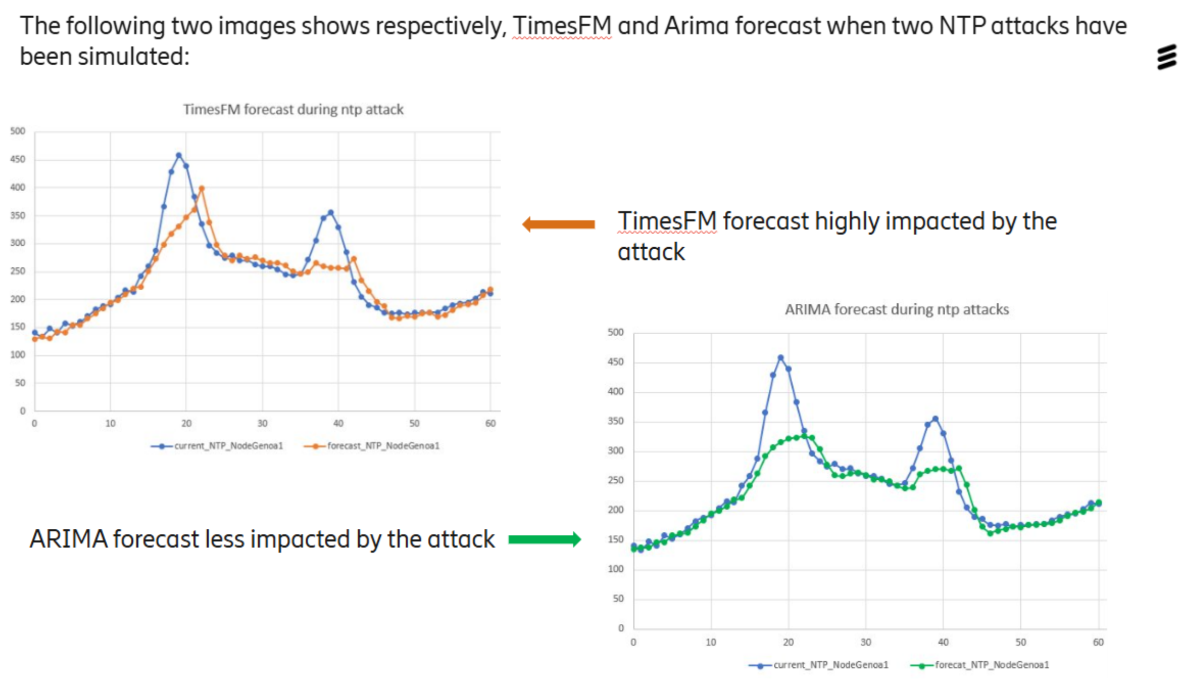
One of the key strengths of DEME's internal architecture lies in its innovative use of a pipelined tree structure. This design allows for the flexible integration and combination of various machine learning (ML) techniques in the search for the most effective forecasting solution.

During the first phase of experimentation, a diverse range of algorithms was tested—including an Ericsson custom one (patent protected [22]), ARIMA, Random Forest, XGBoost, and Logistic Regression—to identify the most promising approach. Among these, ARIMA demonstrated the best performance.

In the second iteration, the focus shifted to comparing ARIMA with TimesFM [23], a more recent and advanced time-series forecasting model developed by Google. TimesFM is a pre-trained model built on extensive real-world time-series data and is capable of generating forecasts for previously unseen datasets with adaptable prediction horizons.

This comparison was part of an exploratory analysis aimed at enhancing the performance of the threat detection component. The evaluation involved head-to-head testing of ARIMA and TimesFM to determine which model delivers more accurate forecasts in the specific context of threat identification.

As illustrated in the following Figure 22: TimesFM vs ARIMA, test results show that while TimesFM represents a significant leap forward in general-purpose forecasting capabilities, within our specific use case of the Threat Detector, ARIMA continues to outperform it in terms of effectiveness (for more details please refer to following Annex A).


The following two images shows respectively, TimesFM and Arima forecast when two NTP attacks have been simulated:

TimesFM forecast highly impacted by the attack

ARIMA forecast less impacted by the attack

Figure 22: TimesFM vs ARIMA

6.3.9 Innovative aspects summarization

In conclusion, focusing on the innovative dimensions of the project, several groundbreaking elements stand out.

- First, the framework integrates for the first time state-of-the-art machine learning technologies within a cybersecurity context alongside equally advanced systems like Digital Twins. This synergy enhances the overall detection and response capabilities, producing more effective outcomes than any individual technology could achieve alone.
- Second, in addressing the critical challenge of minimizing detection time—essential to ensure mitigation actions are still feasible before a network spirals out of control—the system employs fast anomaly detection and learning-based methods. These allow attacks to be identified the moment a deviation from expected behavior occurs, without waiting for predefined thresholds to be crossed.
- Third, in line with this goal, detection sources are strategically selected as close as possible to the root of the potential threat. For example, in the case of NTP DDoS amplification attacks, the system monitors upstream NTP MONLIST queries rather than downstream large response packets, significantly reducing detection latency.
- Fourth, the framework includes both cutting-edge algorithms and custom-developed HORSE algorithms, the latter protected by patents [22], reflecting a strong commitment to technological innovation.

- Fifth, the internal architecture adopts a pipelined tree structure, please see Figure 23, to overcome one of the major limitations in traditional cybersecurity ML applications: siloization. While many commercial solutions claim broad attack coverage, in practice, ML models tend to specialize in specific threat types, operating in isolated silos. This architecture addresses that limitation by deploying multiple specialized detectors in parallel, whose outputs are then aggregated and correlated in downstream stages. The final Egress stage has a unified view of all preceding outputs, enabling it to detect complex or combined attacks that would otherwise go unnoticed by isolated detectors.
- Finally, instead of using a binary on/off attack detection model, this system assigns a risk probability or confidence score to each detection event. This probabilistic approach enables higher-level reasoning and integration with other intelligent modules, potentially powered by complementary technologies, allowing for a more adaptive and intelligent cybersecurity response strategy.

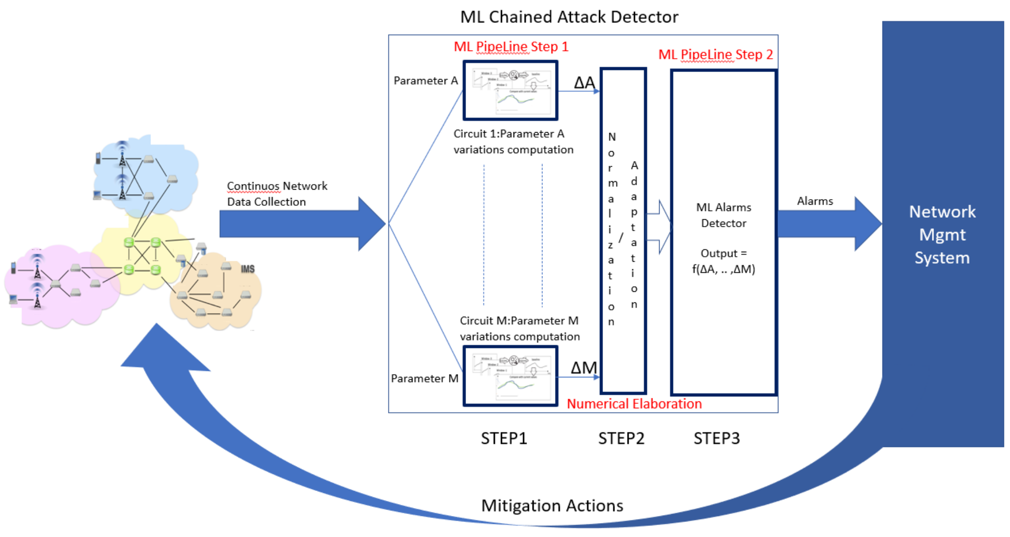


Figure 23: HORSE Threat Detector Innovative Block Diagram

7 Conclusions

This deliverable marks the completion of WP3 (HORSE Platform Intelligence), summarizing the final implementation, refinement, and validation of all the components developed within the five WP3 tasks. The modules—previously released in their IT-1 form—have now reached their final IT-2 stage, incorporating all planned features and improvements, and are ready for full and final integration within the HORSE platform.

The activities conducted in this final phase have focused on testing, debugging, and ensuring consistency with the architectural requirements defined in WP2, while also supporting preliminary integration activities in WP5. As WP3 officially concludes at M30, the outcomes reported in this deliverable represent the definitive state of the Platform Intelligence layer.

Going forward, the emphasis will be on WP5, which is dedicated to the integration, validation, and final release of the complete HORSE platform. The components delivered here are fully aligned and prepared for that next stage, ensuring a smooth transition into final deployment and platform-wide interoperability.

References

[1] Kubernetes https://kubernetes.io/

[2] Kubernetes Network Emulator https://github.com/openconfig/kne

[3] Open5GS - Open Source implementation for 5G Core and EPC, i.e. the core network of LTE/NR network (Release-17) - https://open5gs.org/

[4] Open5GS – A C-language Open Source implementation of 5G Core and EPC - https://github.com/open5gs5

[5] HORSE project consortium, Grant Agreement, August 2022

[6] HORSE project consortium, D2.1 "HORSE Landscape: Technologies, state of the art, AI policies and requirements (IT-1)", June 2023

[7] HORSE project consortium, D2.2 "HORSE Architectural Design (IT-1)", September 2023

[8] HORSE project consortium, D2.3 "HORSE Landscape: Technologies, state of the art, AI policies and requirements (IT-2)", November 2024

[9] HORSE project consortium, D2.4 "HORSE Landscape and Architectural Design", December 2024

[10] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling and F. H. P. Fitzek, "An Open-Source Testbed for Virtualized Communication Networks," in IEEE Communications Magazine, vol. 59, no. 2, pp. 77-83, February 2021, doi: 10.1109/MCOM.001.2000578.

[11] Comnetsemu, https://git.comnets.net/public-repo/comnetsemu

[12] "Mininet website", http://mininet.org

[13] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, Nick McKeown, "Reproducible Network Experiments Using Container-Based Emulation," CoNEXT'12, December 10–13, 2012, Nice, France.

[14] D. Muelas, J. Ramos and J. E. L. d. Vergara, "Assessing the Limits of Mininet-Based Environments for Network Experimentation," in IEEE Network, vol. 32, no. 6, pp. 168-176, November/December 2018, doi: 10.1109/MNET.2018.1700277

[15] HORSE project consortium, D3.1 "HORSE Platform Intelligence developed (IT-1)", September 2023

[16] A. Giannopoulos, S. Spantideas, N. Kapsalis, P. Karkazis and P. Trakadas, "Deep Reinforcement Learning for Energy-Efficient Multi-Channel Transmissions in 5G Cognitive HetNets: Centralized, Decentralized and Transfer Learning Based Solutions," in IEEE Access, vol. 9, pp. 129358-129374, 2021, doi: 10.1109/ACCESS.2021.3113501.

[17] Synthetic Data Set for Network Data Analytics Function (NWDAF), https://github.com/sevgicansalih/nwdaf_data

[18] 5GAD-2022 5G attack detection dataset, https://github.com/IdahoLabResearch/5GAD

[19] 5GC PFCP Intrusion Detection Dataset, https://ieee-dataport.org/documents/5gc-pfcp-intrusion-detection-dataset-0

[20] 5G-NIDD: A Comprehensive Network Intrusion Detection Dataset Generated over 5G Wireless Network, http://ieee-dataport.org/10203

[21] Flower – A Friendly Federated Learning Framework, https://flower.dev/

[22] Gemme Luciano, Cappelli Marco, Ericsson Patent WO2021190760 “Determining an Alarm Condition” https://www.patentguru.com/search?q=WO2021190760

[23] TimesFM - (Time Series Foundation Model) : A pretrained time-series foundation model developed by Google Research for time-series forecasting - https://github.com/google-research/timesfm

[24] Vaadata – “How to strengthen the security of your APIs to counter the most common attacks” - API security, vulnerabilities and common attacks

[25] Mitre – Fight – “Unauthorized access to Network Exposure Function (NEF) via token fraud” - https://fight.mitre.org/techniques/FGT5011/

[26] NestJS – A progressive Node.js framework for building efficient, reliable and scalable server-side applications - https://nestjs.com/

[27] Redis – World’s fastest data platform- https://redis.io/

[28] Bull – Premium Queue package - https://github.com/OptimalBits/bull

[29] R. Bruschi CNIT, Giuseppe Burgarella Ericsson “A Lightweight Prediction Method for Scalable Analytics and Multi-Seasonal KPIs” - https://link.springer.com/chapter/10.1007/978-3-319-67639-5_6

[30] Vaia I. Kontopoulou, Athanasios D. Panagopoulos, Ioannis Kakkos, George K. Matsopoulos “ A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks“ - https://ideas.repec.org/a/gam/jftint/v15y2023i8p255-d1206557.html

[31] “ARIMA based algorithms vs neural networks in anomaly detection” - https://www.eyer.ai/blog/arima-based-algorithms-vs-neural-networks-in-anomaly-detection/

[32] Haitham Fawzy “A Comparative Simulation Study of ARIMA and Computational Intelligent Techniques for Forecasting Time Series Data”, https://www.researchgate.net/publication/357522210_A_Comparative_Simulation_Study_of_ARIMA_and_Computational_Intelligent_Techniques_for_Forecasting_Time_Series_Data

[33] Joao Vitor Matos Goncalves, Michel Alexandre, Michel Alexandre “ARIMA and LSTM: A Comparative Analysis of Financial Time Series Forecasting” - https://ideas.repec.org/p/spa/wpaper/2023wpecon13.html

[34] Time Series Forecasting: ARIMA/VARIMA vs Machine Learning/Deep Learning - https://stats.stackexchange.com/questions/487970/time-series-forecasting-arima-varima-vs-machine-learning-deep-learning

[35] Prometheus – Open Source metrics and monitoring for your systems and services - https://prometheus.io/

Annex A - ARIMA Performance Analysis for Telecommunications Network Parameter Prediction in Cybersecurity Anomaly Detection

ARIMA Performance Characteristics

The ARIMA model demonstrated exceptional performance in our telecommunications network context, exhibiting several key advantages that align with recent academic findings [29]. The model showed remarkable efficiency in terms of computational resources while maintaining high predictive accuracy. Notably, ARIMA achieved optimal performance without requiring extensive historical datasets for training, a significant advantage in operational environments where rapid deployment is essential.

The model's interpretability proved invaluable for understanding the underlying network behavior patterns. The clear statistical meaning of ARIMA's parameters (p, d, q) facilitated both model validation and result interpretation, crucial factors in cybersecurity applications where understanding the reasoning behind predictions is as important as the predictions themselves.

Comparative Analysis with Alternative Algorithms

To validate ARIMA's effectiveness, comprehensive performance comparisons were conducted against several established machine learning algorithms, including Random Forest, XGBoost, and Logistic Regression. Across all evaluation metrics, ARIMA consistently outperformed these alternatives in our specific telecommunications network parameter prediction tasks.

Furthermore, recognizing the emergence of more advanced forecasting technologies, we extended our evaluation to include Google's TimesFM, a state-of-the-art foundation model known for its exceptional performance across diverse forecasting applications. Despite TimesFM's documented superiority in general forecasting scenarios, ARIMA maintained its performance advantage in our specific context of telecommunications network parameters.

Theoretical Context and Literature Alignment

Recent academic literature [30][31][32][33] suggests that while modern AI algorithms generally demonstrate superior performance across most forecasting applications, classical methods like ARIMA maintain competitive advantages in specific scenarios. Our findings strongly support this perspective, particularly the observation that ARIMA excels in scenarios involving univariate time series with clear seasonal patterns and limited computational resources.

The success of ARIMA in our telecommunications context can be attributed to the alignment between the model's assumptions and our data characteristics. The stationarity requirement, often viewed as a limitation, actually represented a strength in our case, as telecommunications network parameters naturally exhibit the stationary behavior (after appropriate transformations) that ARIMA expects.

Implications and Future Research Directions

The superior performance of ARIMA in our specific telecommunications network anomaly detection context demonstrates the continued relevance of classical statistical methods in specialized applications. However, current research trends indicate significant potential in hybrid modeling approaches that combine ARIMA's strengths with modern machine learning capabilities.

Recent studies [34] have shown that hybrid models, such as ARIMA-LSTM combinations, can reduce forecasting errors by 18-35% compared to individual models. This finding suggests that while ARIMA proved optimal for our current telecommunications network parameters, future research iterations should explore hybrid approaches that leverage ARIMA's seasonal modeling capabilities alongside the pattern recognition strengths of modern machine learning algorithms.

Conclusion

This research demonstrates that ARIMA remains a highly effective solution for telecommunications network parameter prediction in cybersecurity anomaly detection contexts, particularly when dealing with multi-seasonal time series data. The model's combination of interpretability, computational efficiency, and robust performance with limited training data makes it an excellent choice for operational cybersecurity systems.

However, the promising results achieved by ARIMA should be viewed as a foundation rather than a conclusion. Future research directions will focus on developing hybrid models that combine ARIMA's proven effectiveness in our specific context with advanced machine learning techniques, potentially achieving even greater performance improvements while maintaining the operational advantages that made ARIMA successful in this telecommunications network security application.

The next experimental phase will therefore concentrate on hybrid solution development, representing a natural evolution from our current findings toward more sophisticated and potentially more effective anomaly detection capabilities for telecommunications network cybersecurity.

Co-funded by the European Union