Smart, Automated, and Reliable Security Service Platform for 6G

Deliverable D3.3
ROBUST-6G Trustworthy and Sustainable AI Prototype

ROBUST-6G project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101139068.

Date of delivery:	30/09/2025	Version:	1.0
Project reference:	101139068	Call:	HORIZON-JU-SNS-2023
Start date of project:	01/01/2024	Duration:	30 months

Document properties:

Document Number:	**D3.3**

Document Title:	ROBUST-6G Trustworthy and Sustainable AI Prototype
Editor(s):	Giovanni Perin (UNIPD), Michele Rossi (UNIPD)
Authors:	Giovanni Perin (UNIPD), Michele Rossi (UNIPD), Bartlomiej Siniarski (UCD), Fernando Torres Vega (UMU), Enrique Tomás Martínez Beltrán (UMU), Manuel Gil Pérez (UMU), Ioannis Pitsiorlas (EUR), Marios Kountouris (EUR), Ömer Faruk Tuna (EBY), Leyli Karaçay (EBY), Betül Güvenç Paltun (EBY), Farah Abed Zadeh (UCD), Eunjeong Jeong (LIU)
Contractual Date of Delivery:	30/09/2025
Dissemination level:	PU[1]/SEN
Status:	Final
Version:	1.0
File Name:	ROBUST-6G D3.3_v1.0

Revision History

Revision	Date	Issued by	Description
0.1	01.07.2025	ROBUST-6G WP3	Initial draft with ToC and assignments
0.2	22.08.2025	ROBUST-6G WP3	First draft of section 2 with prototype architecture
0.3	05.09.2025	ROBUST-6G WP3	First draft of prototype's modules
0.4	10.09.2025	ROBUST-6G WP3	Draft ready for internal review
0.5	20.09.2025	ROBUST-6G-WP3	All comments addressed, document review, written summary, introduction, and conclusions
1.0	24.09.2025	ROBUST-6G-WP3	Document formatted for submission

Abstract

In this deliverable, the ROBUST-6G WP3 team presents the first version of the "Trustworthy and Sustainable AI prototype" and the way in which it interfaces with other layers of the ROBUST-6G architecture.

Specifically, the prototype is composed of four modules: i) the **decentralized federated learning (DFL) framework**, a container-based decentralized learning engine; ii) the **trustworthy AI** module, which integrates privacy, security, and adversarial learning services; iii) the **sustainable AI** module, offering context-based optimization to make training processes energy-efficient and models sustainable by design; iv) the **explainable AI** module, adding explainability services to the prototype. The modules ii)-iv) can be coupled via Distributed Federated Learning (DFL) or used as standalone entities if the requested models are to be trained in a centralized or server-based FL fashion.

The prototype interfaces with external layers (zero-touch security management and network) through dedicated APIs and with the global model repository (GMR), where trained models and relevant training/efficiency metrics are stored. The GMR can also provide model deployment to the users via container-based execution, where all the necessary dependencies are ready to be used during runtime.

[1] SEN = Sensitive, only members of the consortium (including the Commission Services). Limited under the conditions of the Grant Agreement

PU = Public

Keywords

AI/ML prototype, Decentralized learning, Trustworthiness, Sustainability, Explainability, Services integration.

Disclaimer

Executive Summary

This deliverable presents the Robust-6G WP3 prototype. It is an **integrated system** with two main objectives: 1) storing pretrained machine learning/artificial intelligence models along with their attributes, so that they can be consumed on demand by other Robust-6G functional components, and 2) perform model training on-demand (in case a trained model for a certain task is not yet available). For 1), a so-called Global Model Repository (GMR) is created and maintained, where each model has attributes that describe, among others, its input and output data formats and the model performance. For 2), new models can be defined and trained using Decentralized Federated Learning (DFL), which is suitable for distributedly training models in a network via multiple (edge) nodes. In this way, data privacy is preserved as the data used for training will never leave the involved endpoints (here referred to as "federation nodes"), which will use their (local) data to produce model updates. The WP3 DFL module will interact with such endpoints, orchestrating them, for example regulating the frequency of their updates and handling the reception of model updates from each. Different from standard Federated Learning approaches, there is no centralized unit that aggregates the local models, but the federation nodes participate in the learning via a distributed peer-to-peer process, so they will all contribute to the construction of the final model without the need for a central aggregator. The removal of the central aggregator leads to a stronger mechanism, which is resilient to selective attacks to any of the contributing federation nodes. Once a final model is obtained, it is tested, validated and finally exposed, adding it to the GMR. We underline that training can also be performed in a standard centralized fashion, in case the data is centrally available and there is the need for doing so, standard training procedures are applied in this case (e.g., backpropagation for neural networks). It is also worth noting that the adopted training procedures are robust against adversarial attacks, and the training strategy can be selected from a catalogue. The prototype is coded in a modular way, which allows its maintenance and the integration of additional training procedures and model metrics.

The prototype is realized by prioritizing three aspects:

- **Trustworthiness:** this module can leverage the reputation or trust-based metrics from the DFL framework, providing a quantitative assessment of the privacy level of the developed models.
- **Sustainability:** the sustainability module can optimize the energy spent in the decentralized federated learning, trading final model accuracy for energy consumption.
- **Explainability:** this module can generate explainability reports for the models produced by the DFL.

The Robust-6G security management layer and the network layer can query the GMR for **production-ready models**, such as energy optimisers, anomaly detectors, security threat detectors/classifiers or explainability services, etc.

The models stored in the GMR can be bundled with their required execution environment and packaged into a Docker container. This containerised form makes the models portable and ready for deployment across different types of production environments, such as Microsoft Azure, Amazon SageMaker, Databricks, or Kubernetes-based infrastructures. The containerization of modules ensures the following properties:

- **Reproducibility:** each container encapsulates dependencies, libraries, and runtime environments, ensuring that experiments can be reproduced consistently across different setups.
- **Isolation:** components operate in separate containers, preventing conflicts between dependencies and enabling independent scaling of each service.
- **Portability:** containerised components can be deployed seamlessly across different hosts, operating systems, or infrastructures.

In addition, models can be consumed remotely, for example by being served through lightweight frameworks such as Flask or raw models can be also sent, allowing WP4 and WP5 to directly integrate them into their own sub-systems.

The present deliverable D3.3 is organized as follows:

Section 2 reports a discussion of the full prototype, explaining the design principles, the role of its components and how they interact. **Section 3** presents the DFL framework, addressing design principles, the constituting elements, how the framework can be installed, used and integrated into existing systems (API and framework deployment). Moreover, the system and model output metrics are expounded, it is specified how adversarial attacks can be simulated in the training phase to gauge the model ability to counteract them and obtain resilience metrics (to be exposed in the GMR with the other model attributes). A practical guide to use the tool follows and, finally, some details are added on future extensions of the framework. **Sections 4**, **5** and **6** respectively cover the trustworthiness, the sustainability and the explainability of the developed framework and of the models that it generates via either distributed or centralized training. **Section 7** presents our concluding remarks.

Table of Contents

List of Tables

List of Figures

Acronyms and abbreviations

Term	Description
AI	Artificial Intelligence
API	Application Programming Interface
CPCF	Conformal Prediction Confidence Factor
CPU	Central Processing Unit
DFL	Decentralized Federated Learning
FL	Federated Learning
FLCF	Federated Learning Coordination Function
GMR	Global Model Repository
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IDS	Intrusion Detection Systems
IID	Independent, Identically Distributed
ML	Machine Learning
NET	Network Layer
PMF	Privacy Management Function
SHAP	Shapley Additive Explanations
SHIELD	Selective Hidden Input Evaluation for Learning Dynamics
SNN	Spiking Neural Networks
VAE	Variational Autoencoder
WP	Work Package
XAI	Explainable Artificial Intelligence
ZSM	Zero-touch Security Management layer

1 Introduction

1.1 Motivation, objectives and scope

The present deliverable D3.3 presents the WP3 "Trustworthy and Sustainable AI prototype" and the way in which it interfaces with other layers of the ROBUST-6G architecture.

The WP3 prototype is responsible for AI **model training** and **maintenance**, and for **delivering the trained AI models** to the other Robust-6G architectural components.

This is achieved by four modules, namely:

1) the **decentralized federated learning (DFL) framework**, offering several learning primitives such as peer-to-peer (decentralized) federated learning, centralized federated learning or classical model training via a single server.
2) the **trustworthy AI** module, which integrates privacy, security, and adversarial learning services. For instance, dedicated learning solutions were developed to make node updates robust against adversarial attacks.
3) the **sustainable AI** module, offering context-based optimization to make training processes energy-efficient and models sustainable by design. The former is achieved by orchestrating the transmission of model updates in distributed learning processes (e.g., according to residual node energies or energy intake from renewable energy sources), whereas the latter is achieved by applying standard techniques such as weight quantization and pruning or novel approaches developed within Robust-6G that involve spiking neural network designs.
4) the **explainable AI** module, which adds explainability services, by allowing training models with embedded explainability techniques and measures, e.g., measures for output uncertainty and model forgetting.

The modules ii)-iv) can be coupled via Distributed Federated Learning (DFL) or used as standalone entities, in case the requested AI models are to be trained in a *centralized* or *server-based* Federated Learning (FL) fashion.

The prototype interfaces with external layers (zero-touch security management and network) through an API and with the global model repository (GMR), where trained AI models and their attributes are stored. The GMR can additionally provide model deployment to the users via container-based execution, where all the necessary dependencies are ready to be used during runtime.

1.2 Document structure

Section 2 of this deliverable presents at high level the architecture of the first version of the Trustworthy and Sustainable AI prototype, its building blocks and the role of the global model repository (GMR), which, through the prototype's API, delivers models and metrics to the users (other ROBUST-6G layers). Four key sections delve specifically into the services offered by the modules of the prototype, referring to and integrating the previous partner components (D3.2). Specifically, Section 3 presents the decentralized federated learning (DFL) framework, together with a walkthrough on its use and the possible integration of the services of the other prototype modules. Section 4 introduces the Trustworthiness module, Section 5 the Sustainable AI module, and Section 6 the Explainable AI module. These latter three sections explain in detail the services offered and their working flows (with dedicated diagrams), and end with a schematic summary of what relevant information from the specific module is dumped into the GMR to be used or inspected by the users. Conclusions are drawn in Section 7.

2 Description of the prototype architecture

2.1 Introduction

The work in this deliverable presents the Robust-6G WP3 prototype. It addresses three fundamental aspects of 6G systems: **trustworthiness**, **sustainability**, and **explainability**. These aspects are not only central to building secure and resilient future networks, but also essential for enhancing public trust (i.e., users have confidence that sensitive data will not be misused) and ensuring compliance (i.e., network operators being able to demonstrate legal and regulatory auditing, privacy-by-design and explainable AI models). In this Section, we describe a prototype that integrates three dedicated modules, each focusing on one of these aspects. While each module produces distinct outcomes, their common output is the generation of AI/ML model artifacts that can be utilised across the ROBUST-6G architecture. To ensure consistency, accessibility, and transparency, all such models are stored, versioned, and made available through a unified entity: the Global Model Repository (GMR) described in more detail in Section 2.4.

2.2 Trustworthy, Sustainable and Explainable AI Prototype Overview

The WP3 prototype depicted in Figure 2-1 represents the consolidation of a wide range of mechanisms, techniques, and solutions that were developed throughout the earlier phases of this work package to date. In Deliverable 3.2 [ROB24-D32], more than a dozen distinct components were presented, each addressing specific aspects of *trust*, *sustainability*, and *explainability*. These individual contributions demonstrated technical value, and the consortium undertook an integration effort to merge these components into three coherent modules: the **Trustworthiness Module** (detailed in Section 4), the **Sustainability Module** (detailed in Section 5), and the **Explainability Module** (detailed in Section 6). Each of these modules incorporates the best-performing techniques and complementary functionalities from the previously reported components. For example, adversarial robustness methods, privacy-enhancing methods, anomaly detection strategies, and trust metrics have been consolidated into the Trustworthiness Module; energy monitoring, adaptive scaling, and efficiency optimisation mechanisms are embedded in the Sustainability Module; and feature attribution, fidelity measurement, and transparency-enhancing algorithms form the Explainability Module. This process has transformed what were once separate building blocks into an integrated system capable of producing operational models aligned with the objectives of WP3.

As depicted in Figure 2-1, the produced operational models can be requested by both the Zero-touch Security Management (ZSM) layer developed in WP4 and by the Network Layer developed in WP5. In the default path, these layers query the GMR via API.

If a model in the catalogue matches the functional and performance requirements specified in the request—for example, target task (intrusion detection, anomaly classification, compliance check), input data type (RAN telemetry, core logs, edge events), and minimum performance thresholds (e.g., detection accuracy, latency, energy efficiency)—the GMR returns the corresponding model artifact immediately.

If no model satisfies the stated requirements, a new training process is triggered. The requirements are forwarded to one of the three WP3 modules—Trustworthiness, Sustainability, and Explainability—which will train a new model using the data and features exposed from the lower layers. Training can proceed either with the optional Distributed Federated Learning (DFL) framework or with non-DFL pipelines. In both cases, the resulting model artifact is registered back into the GMR and made available to the requesting layer.

The result is a streamlined prototype that is technically robust and architecturally coherent, demonstrating that the knowledge and outputs generated in previous deliverables have been unified into a small number of consolidated, impactful modules.

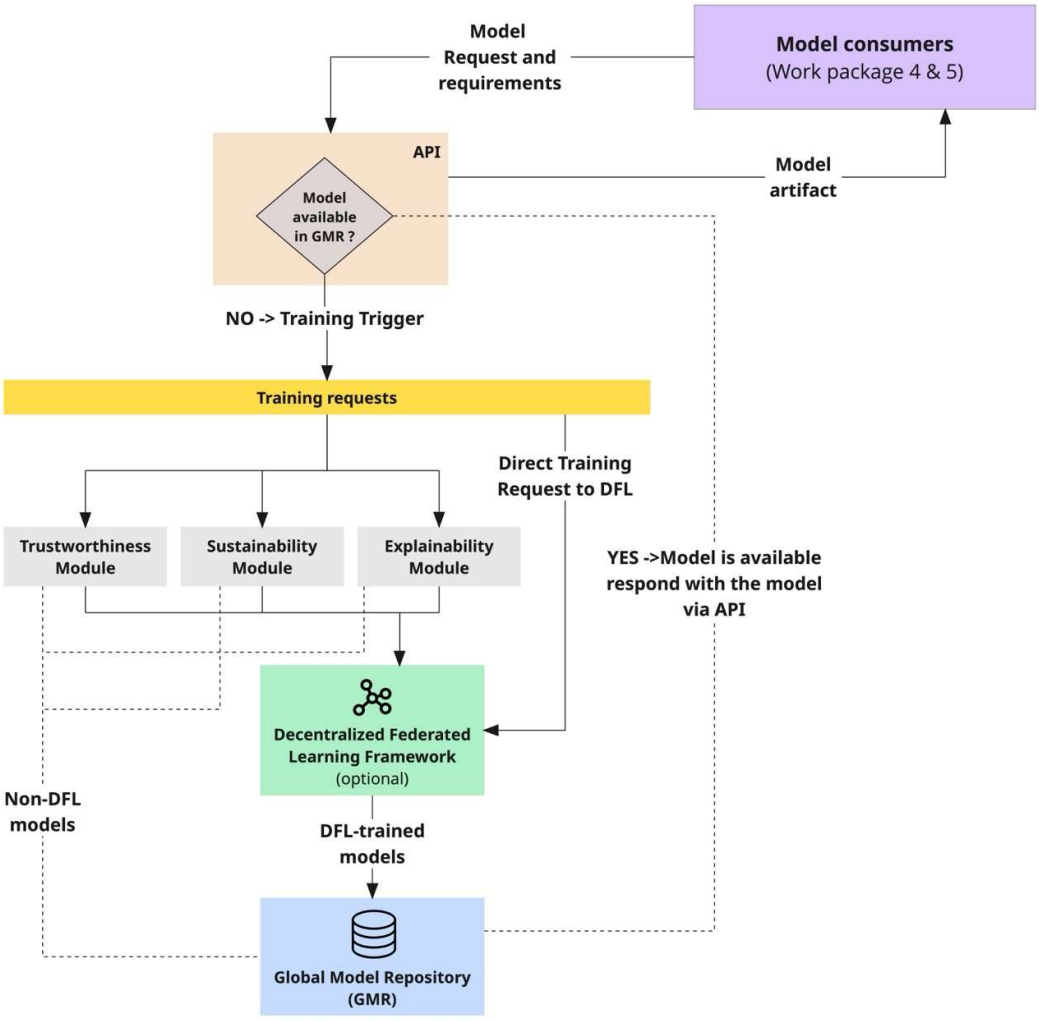

Figure 2-1 WP3 Prototype Overview

Our prototype, depicted in Figure 2-1, can be presented through the architectural lens by mapping it directly onto the architectural view defined for our work package. The architectural view of WP3 shown in Figure 2-2 provides the structural framework, showing both the internal components (intra-WP modules and interfaces) and the external links to other work packages.

Our prototype concretizes this view from two different perspectives:

- **Intra-WP perspective:** It demonstrates how the core functional blocks of our work package (Trustworthy and Sustainable AI services) are implemented and integrated, following the logical layering and data flows defined in the architecture. This highlights the progress from conceptual design to working instantiation. Here, we see the interaction of Trustworthy AI services layer with the AI services management layer to enable all the technical contributions.
- **Inter-WP perspective:** It illustrates how our implementation exposes the interfaces and dependencies toward other work packages. These connections make explicit how our prototype contributes to the overall system architecture of the project, ensuring alignment and interoperability across WPs.

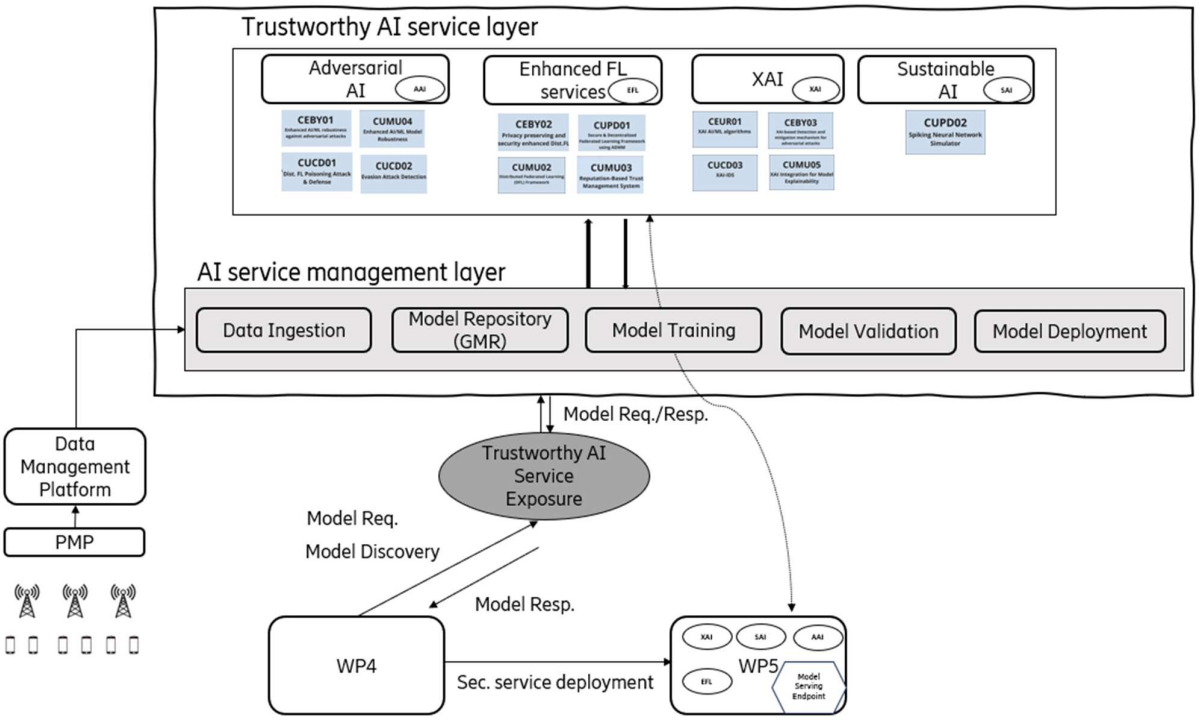

Figure 2-2 WP3 Architectural view

2.2.1 Decentralized Federated Learning (DFL) in the Prototype

The DFL Framework is an important building block of the prototype in scenarios where decentralisation, privacy, and resilience are required. Unlike classical federated learning, DFL eliminates the need for a central aggregator (server) and instead coordinates model training across nodes in a peer-to-peer fashion. It produces a variety of artefacts, including node-specific models, round-based snapshots, and a final global model after convergence.

These artefacts are highly relevant for WP3. For example, the Trustworthiness Module can leverage reputation scores or trust-based metrics from the DFL Framework, while the Sustainability Module can optimise the energy-aware training of models through the DFL Framework and evaluate efficiency scores for the trained models. Similarly, the Explainability Module can generate interpretability artefacts for models produced in decentralized training settings. All outputs, whether produced through DFL or by traditional methods, are captured and preserved in the GMR, ensuring a unified registry and traceability of model artefacts. The actual model lifecycle (including training, updating, and retraining) is managed by the respective modules, while the GMR provides a consistent access point and version-controlled storage. The DFL is described in fine details in Section 3.

2.2.2 Global Model Repository: Concept, Interfaces and Access

2.2.2.1 Concept and contents

The GMR is envisaged as a central facility for storing, cataloguing, and versioning the model artefacts produced within WP3. It acts as a structured repository that preserves not only the models themselves but also their runtimes, libraries, and dependencies, ensuring that they can be reproduced and executed consistently at later stages. Alongside the models, the GMR maintains rich metadata describing their provenance, such as the datasets used for training, performance metrics, and contextual indicators including fairness, robustness, or energy efficiency. Models can be annotated with statuses such as experimental, staging, or production, providing a clear view of their maturity and readiness without the GMR itself managing deployment, retraining, or other aspects of the full model lifecycle.

2.2.2.2 Access and Interfaces

As illustrated in Figure 2-3, models stored in the GMR can be bundled with their required execution environment and packaged into a Docker container. This containerised form makes the models portable and ready for deployment across different types of production environments, such as Microsoft Azure, Amazon SageMaker, Databricks, or Kubernetes-based infrastructures. In addition, models can be consumed remotely, for example by being served through lightweight frameworks such as Flask for testing and demonstration purposes, or by supporting batch prediction workloads. Within ROBUST-6G, a third option has also been implemented: in addition to containerised artefacts, the GMR can directly provide raw model artefacts, enabling WP4 and WP5 to integrate and deploy them according to their specific pipelines. In this way, the GMR acts as a single reference point for all WP3 models, ensuring that they can be reliably reused, updated, and consumed across different network layers.

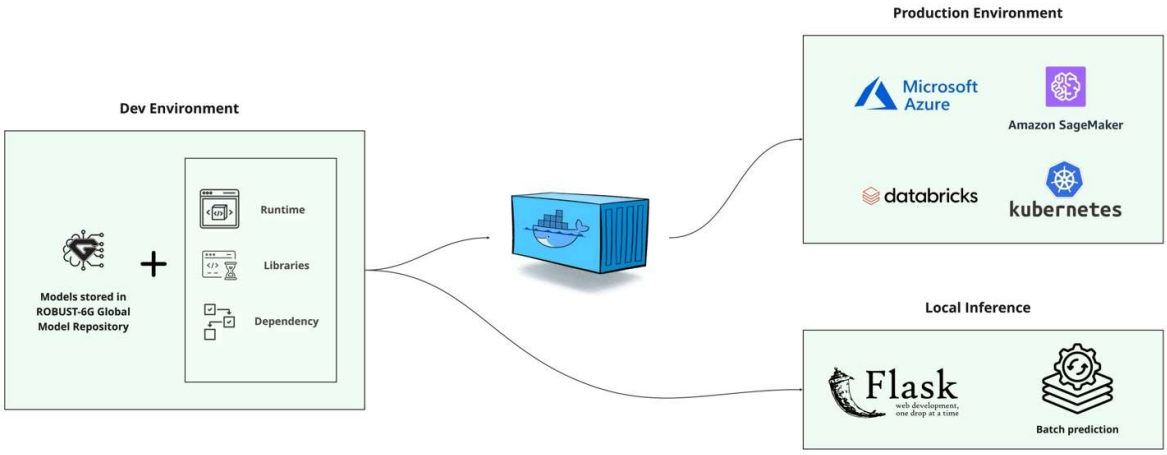

Figure 2-3 The role of GMR in deployment process

The GMR is a critical enabler for the other layers of the project architecture. The security management layer and the network layer can query the GMR for production-ready models, such as energy optimisers, anomaly detectors, or explainability services. The security orchestrator, like other consumers, can request the integration of trustworthiness or sustainability models into its decision-making pipelines through the AI prototype, while models for tasks such as anomaly or threat detection can be retrieved in the same way. At the physical layer, including edge devices, lightweight versions of models can be provided in formats optimised for resource-constrained environments. Models are accessible through stable interfaces, such as queries by name, version, or alias as shown in Table 2-1.

Event-driven mechanisms also allow the GMR to notify consumers when a new model is promoted to production. This ensures that orchestration and deployment can be performed in near-real time, without manual intervention.

Table 2-1 Core REST Endpoints for Managing Registered Models and Versions

REST method	Endpoint	Description
POST	registered-models/create	Create a new registered model
GET	registered-models/get	Get details of a registered model
GET	registered-models/list	List all registered models
POST	registered-models/delete	Delete a registered model
POST	model-versions/create	Create a new version of a model
GET	model-versions/get	Get details of a specific model version
GET	model-versions/search	List (search) all versions of a model
POST	model-versions/delete	Delete a specific model version

2.3 Summary

The WP3 prototype demonstrates the integration of three complementary modules—**trustworthiness**, **sustainability**, and **explainability**—into a coherent system. Some of the functionalities offered by these modules interact with the DFL Framework, while others enhance traditional training methods. In both cases, the outputs are unified within the GMR, which ensures consistency, accessibility, and transparency. By cataloguing and providing models together with their metadata, the GMR offers a trusted mechanism to deliver WP3 outputs to the management and orchestration layer as well as the physical layer, thus enabling explainable, sustainable, and trustworthy AI for 6G.

3 Core Decentralized Federated Learning framework

3.1 Introduction

The UMU Core DFL framework provides the essential infrastructure for orchestrating collaborative training across distributed nodes without relying on a centralized server [MB+23]. The framework emphasizes modularity, interoperability, and reproducibility, enabling the design of complex scenarios that include both benign and adversarial behaviours. It is designed to be flexible enough for research purposes while maintaining the scalability required for real-world applications.

3.2 Agnostic Design

The framework has been designed following an **agnostic architecture**, which can be seen outlined in Figure 3-1. This approach allows each module to evolve independently while guaranteeing seamless integration across the whole system. The modular structure promotes extensibility, making it possible to incorporate new functionalities such as datasets, AI/ML models, and adversarial attacks without altering the core logic.

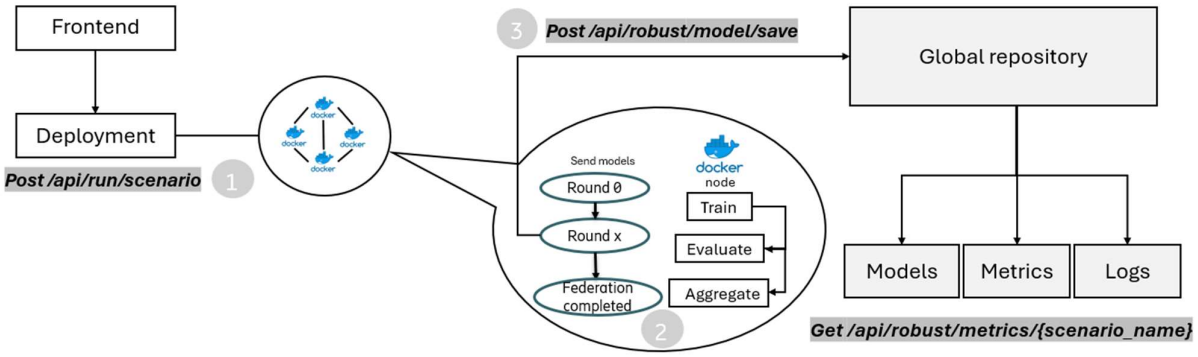

Figure 3-1 DFL Framework architecture

By adopting this design principle, the framework remains highly adaptable to diverse contexts and use cases, facilitating integration with heterogeneous infrastructures and fostering component reusability. Furthermore, communication between modules relies on REST APIs, which reinforces technology independence and simplifies interoperability with external systems.

3.2.1 Frontend

The Frontend constitutes the primary entry point for configuring, executing, and monitoring DFL experiments. The interface is structured into three main areas: Scenario Configuration, Node Configuration, and Metrics Dashboard.

- **Scenario Configuration:** This panel allows the user to define the global parameters of an experiment. Configuration options include the network topology (e.g., fully connected, ring, or random graphs), dataset selection (e.g., MNIST, TON_IoT), data distribution mode (IID or non-IID), model architecture (e.g., MLP, CyberNet), and aggregation algorithm (e.g., FedAvg). Users can also specify

the number of communication rounds and local training epochs, which control the duration and granularity of the experiment. These options ensure that scenarios can be flexibly adapted to a wide range of research objectives, from benchmarking to adversarial robustness testing.

- **Node Configuration**: Each node participating in the federation can be individually configured through the Frontend. Users can assign roles (e.g., aggregator, standard participant), define attack behaviours (e.g., model poisoning, label flipping, or no attack), and specify parameters such as attack strength or the percentage of data manipulated. Nodes can be added or removed dynamically, providing a flexible mechanism to scale experiments and to study heterogeneous behaviours in federated settings. Furthermore, start/stop flags enable fine-grained control over node participation, thereby facilitating scenario customisation and iterative testing.
- **Metrics Dashboard**: The dashboard provides real-time feedback on the state of the experiment. While inactive experiments display a placeholder notification, once executed the dashboard is populated with system- and model-level metrics. These include CPU utilisation, Bytes sent, and F1 Score. This integrated monitoring capability enables operational oversight and scientific analysis, supporting reproducibility and comparability of results.

From a technical perspective, the Frontend communicates with the Controller (explained in Section 3.2.2) and other backend components via REST APIs, ensuring seamless orchestration of experiment lifecycles. A screenshot of this Frontend for a given configuration and results can be shown in Figure 3-2.

3.2.2 Controller

The Controller is the central coordination component of the DFL Framework, this being a key element contained in the Deployment item in Figure 3-1. The Controller is responsible for managing the full lifecycle of experiments, from their initial setup to the completion of training and evaluation. By acting as the orchestrator of distributed processes, the Controller ensures coherence, reproducibility, and robustness across all participating nodes.

Its functionalities can be grouped into three main domains: Initialization, Configuration Management and Orchestration, and Interaction with the GMR.

- **Initialization of Experiments**: The Controller initiates experiments based on configurations submitted either through the graphical Frontend or via the REST API. During this phase, it validates experiment parameters, establishes the network topology, and prepares communication channels between nodes. In scenarios where adversarial behaviour is specified, the Controller ensures that malicious and benign roles are properly instantiated according to the experiment design.
- **Configuration Management and Orchestration**: The Controller parses configuration files and translates them into executable instructions for nodes. This includes dataset allocation, model instantiation, and attack parameterisation. Beyond basic orchestration, it monitors node status during execution and handles potential failures. Such orchestration capabilities make the Controller a resilient backbone of the framework, capable of supporting both static and adaptive experimental designs.
- **Interaction with the GMR**: The Controller ensures consistency across participants by tightly integrating with the GMR. At the start of an experiment, it retrieves the baseline global model, which is distributed to all nodes. After each communication round, it collects node updates, coordinates their exchange among nodes, and ensures that the aggregation results performed by the peers are registered in the GMR. This interaction guarantees reproducibility, as all intermediate versions are tracked, and enables comparative analysis across different runs.

From a technical perspective, the Controller exposes **REST API endpoints** to support external interaction and automation. It can receive experiment definitions, trigger execution, and provide status updates or results to external monitoring systems. This makes it a pivotal integration point for the broader project architecture, allowing seamless interoperability with higher-layer orchestration modules.

3.2.3 Federation Nodes

Nodes represent the core distributed entities of the federation, embodying the decentralized and privacy-preserving principles of the DFL Framework. Each Node is an autonomous participant that contributes to the collaborative training process by performing local computations on its private dataset, thereby ensuring that raw data never leaves the local environment [MB+23]. This design enhances privacy, resilience, and

compliance with data protection requirements, while enabling scalable experimentation across heterogeneous settings.

Each Node performs the following key functions:

- **Local Training**: Nodes independently train a local instance of the model using their own private datasets and supported ML frameworks such as PyTorch. In practical terms, for the execution of any test with the DFL Framework, a partition class distribution with a normalized number of samples, following a non-IID approach, is carried out between the different Nodes of the federation [MSL+24]. Local training can be configured with adjustable parameters such as number of epochs, batch size, and learning rate, enabling fine-grained experimentation and adaptation to diverse data characteristics.
- **Model Update Generation**: Upon completion of local training, Nodes produce model updates instead of sharing raw data. These updates encapsulate the local learning progress and are subsequently returned to the system for aggregation.
- **Behavioural Configurations**: The framework supports heterogeneous Node behaviours, allowing the simulation of both benign and adversarial participants; namely:
    - *Benign behaviour*: Nodes contribute to genuine updates aimed at improving the global ML model's performance.
    - *Malicious behaviour*: Nodes can be configured to emulate adversarial strategies such as model poisoning or label flipping. These capabilities enable systematic evaluation of robustness against security threats.
- **Interaction with the Controller and the GMR**:

  Nodes interact with the Controller (described in Section 3.2.2) at different stages of the experiment. After each training round, Nodes persist their local model updates. The Controller retrieves these stored versions and pushes them to the GMR (introduced in Section 2.2.2 and described in Section 3.2.4). This cyclical process ensures alignment between distributed participants and maintains traceability through repository versioning.

From a deployment perspective, each Node is encapsulated within a containerised environment, guaranteeing reproducibility and isolation of dependencies.

3.2.4 DFL Framework interaction with the GMR

The GMR constitutes the backbone of the DFL Framework's persistence, consistency, and reproducibility mechanisms. This primary entity for sharing information related to the DFL Framework was initially described in Section 2.2.2. It provides a unified facility for storing, managing, and versioning all model artefacts generated during decentralized training.

At the first development stage, the GMR was implemented as a local service, enabling experiments to be executed in contained environments with reproducible conditions. This local implementation ensured that model versioning, storage, and retrieval can be validated and demonstrated reliably during the prototype phase. New releases of the DFL Framework comply with the requirements and tools presented in Section 2.2.2.

Maintaining information in the GMR serves several critical roles and functions within the system:

- **Persistence and Versioning**: All models produced during training are stored within the repository. Each model is versioned and accompanied by metadata such as training configuration, dataset provenance, and performance indicators. This versioning capability ensures that experiments are reproducible and traceable, and that any user can roll back to earlier checkpoints for debugging or comparative analysis.
- **Lifecycle Management**: The repository manages the entire lifecycle of models within the framework. This structured approach allows systematic evaluation of model performance over time and facilitates transparent documentation of results.
- **Integration with the Controller**: The DFL Framework communicates with the repository through the Controller. After each training round, Nodes persist their local model updates in the system. The Controller retrieves these stored models and pushes them to the GMR.

- **Reproducibility and Transparency**: By maintaining historical checkpoints and detailed metadata, the repository ensures that experiments are not only repeatable but also auditable. Users can reproduce past results, evaluate model evolution across rounds, or conduct targeted analysis on specific versions. This increases the reliability of experimental results and helps ensure compliance with reproducibility standards in scientific research.
- **Scalability and Future Deployment**: While the current implementation runs locally, the repository has been **designed for extensibility**; in later stages, deployed across diverse environments such as cloud-based or edge infrastructures. This portability ensures that models generated within the DFL Framework can be seamlessly reused in different contexts without compromising integrity.

3.2.5 Interfacing with external processes/resources

The DFL Framework is designed to operate not only as a standalone experimentation environment but also as an interoperable platform that can be seamlessly integrated with external tools, orchestration systems, and research pipelines. To achieve this, the framework provides well-defined interfaces that enable external processes to configure experiments, monitor their execution, and retrieve results in a programmatic and reproducible manner.

This interoperability is a cornerstone of the framework's design philosophy, ensuring that it can support diverse use cases such as:

- Integration with higher-level orchestration layers developed in other work packages.
- Connection to data pipelines for automated dataset preparation and distribution.
- Support for continuous evaluation workflows, where experiments are triggered and analysed automatically.
- Interfacing with visualisation dashboards or third-party monitoring tools for extended analytics.

3.2.5.1 API endpoints

The primary mechanism for external interaction is a set of RESTful API endpoints exposed by the DFL Framework. These endpoints provide full lifecycle support for experimentation and monitoring, including:

- **Experiment management** – creation, initialization, execution, and termination of experiments.
- **Configuration submission** – uploading of JSON configuration files, either generated manually or by external automation tools.
- **Progress monitoring** – real-time access to system metrics and model metrics.
- **Result retrieval** – download of logs, metrics reports, or final model artefacts for external validation, benchmarking, or further processing.

The use of REST APIs ensures that external processes can interact with the framework in an agnostic manner, relying on widely adopted standards. In its current implementation, the API endpoints operate in a local environment, allowing reproducible and controlled experimentation. However, the architecture anticipates extension to distributed deployments, enabling remote access and integration with large-scale infrastructures in future iterations.

Table 3-1 summarizes the available API endpoints, including their methods, paths, descriptions, expected responses, and error codes. This provides a clear overview of how external components or users can interact with the framework in practice.

Table 3-1 API endpoints

Method	Path	Description	Success Response (200)	Error Codes
GET	/api	Health/landing endpoint that returns a static welcome message.	{"message": "Welcome to the ROBUST Controller API"}	404 Not Found (only if requesting an undefined endpoint outside the API)

POST	/api/robust/run/scenario	Starts a ROBUST scenario: saves the scenario JSON to disk, generates config files, and launches the run.	{"message": "Scenario running successfully", "scenario": "<date>"}	409 if a scenario is already running. 500 on unexpected errors (implementation currently returns 200 with an `error` payload).
POST	/api/robust/stop/scenario	Stops the currently running scenario by terminating dashboard and participant processes.	{"message": "Scenario stopped successfully"}	500 on unexpected errors (implementation currently returns 200 with an `error` payload).
GET	/api/robust/status	Checks whether a ROBUST scenario is running. Returns true if any matching container is running.	{"running": <bool>}	200 with running=false when Docker is missing or command fails (logged as error).
GET	/api/robust/logs	Packages all `.log` files found under `robust/logs` into a ZIP and returns it.	Binary ZIP file. Filename format: `logs_<YYYY-MM-DD_HH-MM-SS>.zip`	500 on unexpected errors.
GET	/api/robust/model/{scenario_name}	Returns model file(s) (.pth) for a scenario. Can filter by participant, by specific round, or pick the best model by metric.	If `round` given: single `.pth` file. If `metric` given: best `.pth` by metric. If neither: ZIP with all models (or all for the participant).	400 invalid metric; 404 when scenario/logs/models not found or no records for participant/round.
GET	/api/robust/metrics/{scenario_name}/	Returns a ZIP containing `metrics.csv` files for each participant across sections: metrics, train, val, test.	Binary ZIP file named `<scenario_name>_metrics.zip`.	404 if scenario logs directory does not exist; 500 on unexpected errors.

3.3 DFL Metrics

The monitoring and evaluation of metrics is a critical functionality of the DFL Framework, as it enables both operational oversight and scientific validation of experimental outcomes. Metrics are systematically collected and reported in the GMR during the execution of experiments to provide insights into system performance, model behaviour, and the impact of adversarial strategies. This dual focus ensures that the framework can be assessed both from a technical infrastructure perspective and from an ML effectiveness perspective.

3.3.1 System metrics

System metrics are focused on monitoring the utilisation of resources and the stability of the underlying infrastructure. These measurements allow the user to evaluate the scalability and efficiency of the framework under diverse workloads and deployment conditions. The following indicators are collected:

- **Resources/CPU_percent** – percentage of CPU utilisation during training and communication.

- **Resources/CPU_temp** – temperature of the CPU, relevant for detecting potential overheating in prolonged or resource-intensive experiments.
- **Resources/RAM_percent** – percentage of RAM usage, particularly important when handling large datasets or models.
- **Resources/Disk_percent** – disk storage utilisation, providing insights into the persistence overhead of local checkpoints and logs.
- **Resources/Bytes_sent** – total number of bytes transmitted over the network by a Node.
- **Resources/Bytes_recv** – total number of bytes received over the network by a Node.
- **Resources/Packets_sent** – number of network packets transmitted, useful for fine-grained network traffic analysis.
- **Resources/Packets_recv** – number of network packets received.
- **Resources/Uptime** – operational duration of a Node or experiment, relevant for assessing stability and long-running scenarios.

Together, these metrics provide a comprehensive view of the computational load, thermal stability, memory and storage efficiency, and communication overhead of the system.

3.3.2 Model metrics

Model metrics are used to track the performance of the trained models across the three standard phases of ML pipelines: training, validation, and testing. These indicators not only measure predictive accuracy but also provide insights into convergence behaviour and robustness. The metrics include:

- **Loss (Train/Test/Validation)** – monitors the optimisation process and provides a direct indicator of learning progression.
- **Accuracy (Train/Test/Validation)** – evaluates the overall proportion of correctly classified samples.
- **Precision (Train/Test/Validation)** – measures the proportion of correctly identified positive samples.
- **Recall (Train/Test/Validation)** – quantifies the proportion of actual positives that were correctly identified.
- **F1Score (Train/Test/Validation)** – harmonic mean of precision and recall, providing a balanced view of model performance.

In addition, the framework collects epoch-based metrics, which capture the evolution of the model's behaviour over time:

- **TestEpoch/Accuracy**, **TestEpoch/Precision**, **TestEpoch/Recall**, **TestEpoch/F1Score** – track the performance of the model at each epoch, supporting fine-grained analysis of convergence dynamics.
- **epoch** – logs the current training epoch, enabling alignment of performance metrics with the temporal progression of the experiment.

These model metrics ensure that experiments can be analysed from multiple perspectives, including global performance, epoch-level dynamics, and phase-specific validation.

3.4 Configuration syntax

The DFL Framework provides a flexible and reproducible configuration mechanism that allows defining, launching, and repeating experiments with minimal overhead. Configurations can be created through different modalities.

3.4.1 Frontend

Through the GUI shown in Figure 3-2, users can configure scenarios interactively, without requiring programming knowledge.

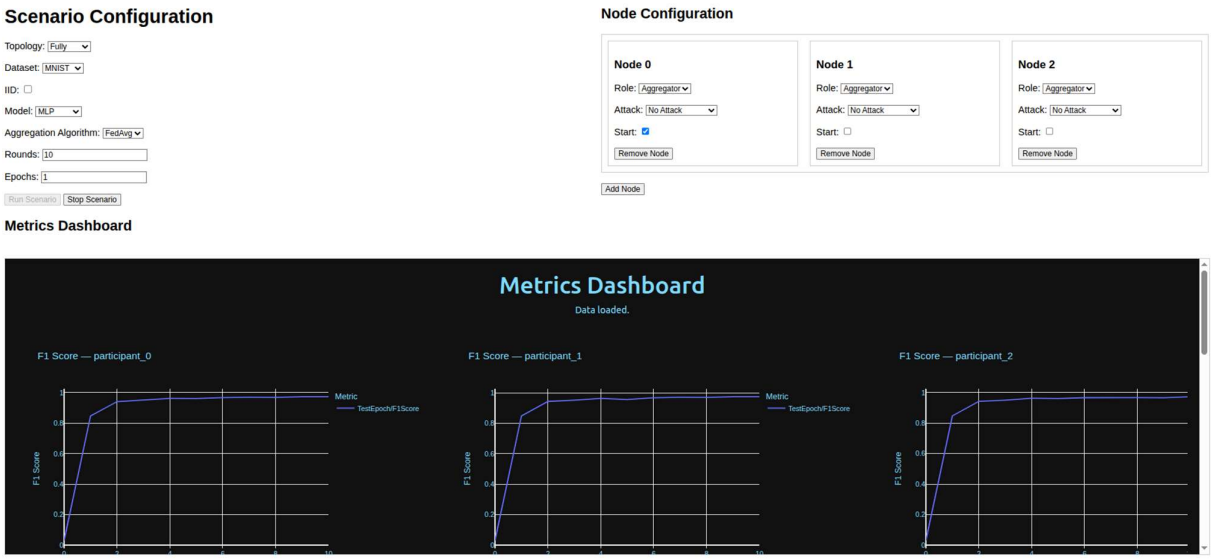

Figure 3-2 Configuration via a Graphical User Interface (GUI)

The Frontend exposes configurable parameters that cover the most relevant aspects of experimental design, including:

- **Node behaviour** – specification of benign or malicious roles, including adversarial strategies such as poisoning or backdoor attacks. This behaviour is implicit in the attack strategy (see below) shown in Figure 3-2 in the Configuration block of each Node. In the example in the figure, the behaviour is benign since *No Attack* is established for that given configuration. Other types of malicious activities will be integrated into the current version of the DFL Framework and incorporated into the Frontend for selection and configuration.
- **Datasets** – selection of training data (e.g., MNIST, TON_IoT) and the distribution scheme (IID vs. non-IID).
- **Models** – definition of model architectures (e.g., MLP, CyberNeT) and training hyperparameters such as batch size, learning rate, and number of epochs.
- **Attack strategies** – configuration of malicious parameters, including attack strength, proportion of compromised Nodes, and targeted behaviours.

3.4.2 JSON via API

For reproducibility and integration with external workflows, the framework also supports configuration via JSON submitted through the REST API. This approach provides several advantages:

- **Reproducibility** – configurations can be exported, stored, and re-executed at any time, ensuring that experiments can be faithfully reproduced.
- **Automation** – external pipelines or tools can dynamically generate JSON configurations and trigger experiments automatically, supporting large-scale or batch experimentation.
- **Integration** – APIs enable seamless interaction with external platforms, such as orchestration layers, CI/CD pipelines, or benchmarking suites.

By using JSON configurations, the framework ensures that experimental designs are transparent, shareable, and scriptable, which is key for collaborative research.

3.4.2.1 Templates (Benign/Malicious Scenario)

To accelerate experimentation and reduce setup complexity, the framework includes a library of predefined templates. These templates capture common experimental configurations and can be instantiated via API. Examples include:

- **Benign scenarios** – standard federated learning settings where all Nodes behave correctly, contributing genuine updates to the global model.

  Note: For readability, benign Nodes are represented without adversarial parameters. In the real framework implementation, these fields are still present in the configuration schema with default values set to "no attack".

```
{
    "topology": "Fully",
    "nodes": {
        "0": {
            "id": 0,
            "ip": "192.168.50.2",
            "port": "45000",
            "role": "aggregator",      # Aggregator (global model consistency) or proxy
            "attack": "No Attack",     # No application of adversarial attacks
            "start": true,
            "adversarial_args": {      # No adversarial attacks
            }
        },
        "1": {
            "id": 1,
            "ip": "192.168.50.3",
            "port": "45000",
            "role": "aggregator",
            "attack": "No Attack",
            "start": false,
            "adversarial_args": {
            }
        },
        "2": {
            "id": 2,
            "ip": "192.168.50.4",
            "port": "45000",
            "role": "aggregator",
            "attack": "No Attack",
            "start": false,
            "adversarial_args": {
            }
        }
    },
    "n_nodes": 3,
    "dataset": "MNIST",                # Dataset to use (MNIST and TON_IoT supported)
    "iid": false,                      # Non-IID selected: each node has a biased dataset
    "model": "MLP",                    # Multilayer perceptron
    "agg_algorithm": "FedAvg",         # Federated Averaging algorithm
    "rounds": "10",
    "accelerator": "cpu",
    "network_subnet": "192.168.50.0/24",
    "network_gateway": "192.168.50.1",
    "epochs": "1",
    "date": "04_09_2025_10_49_10"
}
```

- **Adversarial scenarios** – federations where a subset of Nodes is configured to behave maliciously (e.g., performing data or model poisoning). The specific configuration for determining these types of scenarios is highlighted in red.

```
{
    "topology": "Fully",
    "nodes": {
        "0": {
            "id": 0,
            "ip": "192.168.50.2",
            "port": "45000",
            "role": "aggregator",
            "attack": "No Attack",
            "start": true,
            "adversarial_args": {
            }
        },
        "1": {
```

```
            "id": 1,
            "ip": "192.168.50.3",
            "port": "45000",
            "role": "aggregator",
            "attack": "Model Poisoning",
            "start": false,
            "adversarial_args": {
                "poisoned_sample_percent": 50,
                "targeted": false,
                "target_label": 4,
                "target_changed_label": 7,
                "strength": 10000,
                "perc": 1
            }
        },
        "2": {
            "id": 2,
            "ip": "192.168.50.4",
            "port": "45000",
            "role": "aggregator",
            "attack": "No Attack",
            "start": false,
            "adversarial_args": {
            }
        }
    },
    "n_nodes": 3,
    "dataset": "MNIST",
    "iid": false,
    "model": "MLP",
    "agg_algorithm": "FedAvg",
    "rounds": "10",
    "accelerator": "cpu",
    "network_subnet": "192.168.50.0/24",
    "network_gateway": "192.168.50.1",
    "epochs": "1",
    "date": "04_09_2025_10_46_15"
}
```


Templates serve as blueprints that can be customised further, enabling to any user to quickly explore baseline scenarios before introducing more complex variations.

3.5 Deployment of the DFL infrastructure

The deployment and infrastructure design of the DFL Framework ensure that experiments are executed in a reproducible, scalable, and portable manner. By leveraging containerisation technologies and standard orchestration tools, the framework can operate consistently across heterogeneous environments, from local workstations to distributed clusters. The current implementation focuses on local deployments to facilitate development, validation, and controlled experimentation, while the architecture anticipates future extensions to cloud and large-scale infrastructures.

3.5.1 Containerization with Docker

All core components of the system, including the Frontend, Controller, and Nodes, are encapsulated within Docker containers. This approach provides several advantages:

- **Reproducibility** – each container encapsulates dependencies, libraries, and runtime environments, ensuring that experiments can be reproduced consistently across setups.
- **Isolation** – components operate in separate containers, preventing conflicts between dependencies and enabling independent scaling of each service.
- **Portability** – containerised components can be deployed seamlessly across different hosts, operating systems, or infrastructures.

This container-based architecture guarantees that the framework remains lightweight, flexible, and extensible, supporting both research and operational experimentation. Moreover, communication between containers is managed through a dedicated virtual network, enabling controlled and secure data exchange between Nodes

and the Controller. This design ensures that network traffic within experiments remains isolated from external processes, improving both stability and reproducibility.

The framework also incorporates resource management capabilities, including scalability across machines, allowing containers to be distributed over multiple hosts when larger federations or more demanding experiments are required; and flexible topologies, where network connections between Nodes can be configured to simulate different federation structures (e.g., fully connected, ring, hierarchical). This flexibility allows any type of entity to experiment under diverse conditions, including heterogeneous environments that reflect real-world scenarios.

3.5.2 Installation guide

The installation process of the DFL Framework has been designed to be simple, reproducible, and consistent across environments. By following a sequence of well-defined steps, any partner can set up the local development environment and deploy the system in a controlled manner. The procedure consists of the following stages:

- Virtual environment creation – a Python virtual environment is created to isolate the project dependencies from the host system, ensuring reproducibility and preventing conflicts.

```
python -m venv robust-venv
```

- Environment activation – the virtual environment is activated so that subsequent commands are executed within its isolated context.

```
source robust-venv/bin/activate
```

- Repository cloning – the framework is retrieved from the official repository using a secure URL provided to project partners.

```
git clone https://github.com/CyberDataLab/ROBUST-6G_DFL_Framework.git
```

Note: A token is necessary to download the framework. It was shared with the rest of the partners.

- Project navigation – the user navigates into the cloned project directory, preparing the workspace for installation.

```
cd ROBUST-6G_DFL_Framework
```

- Dependency installation – all required Python libraries and packages are installed from the `requirements.txt` file to guarantee compatibility with the framework.

```
pip install -r requirements.txt
```

- Docker image building – a dedicated Docker image is created to encapsulate the application and its runtime, ensuring portability and consistency.

```
docker build -t robust .
```

- Application execution – the development server is launched with Uvicorn, enabling immediate local interaction with the API and other framework components.

```
uvicorn main:app --reload
```

Upon completion of these steps, the DFL Framework will be available locally and accessible through the following URL: **http://127.0.0.1:8000**

3.6 Usage examples (walkthrough)

This section illustrates how the DFL Framework can be used in practice to train, fine-tune, and deploy ML models. Two complementary interaction modalities are supported: through the graphical frontend and via the REST API/terminal, enabling both intuitive experimentation and automated workflows.

3.6.1 Training an ML Model

Below are two ways to train an ML model.

- **Via the frontend**: Users can create a new scenario directly from the GUI by selecting a dataset (e.g., TON_IoT), specifying the model architecture (e.g., CyberNet), and configuring parameters such as number of rounds, epochs, and node behaviors (benign or adversarial), among others. Figure 3-2 shows an example of use. Once the experiment is launched, the system orchestrates the training process from scratch, and results are displayed in real time through the monitoring dashboard, including system metrics (CPU and bytes sent) and model performance metrics (F1 Score).
- **Via the API/Terminal**: Experiments can also be launched by submitting a JSON configuration through the REST API. For example:

```
curl -X POST http://localhost:8000/api/robust/run/scenario \
    -H "Content-Type: application/json" \
    -d @scenario.json
```

  where *scenario.json* defines the JSON file defining the federation topology, the node configurations (including benign or malicious behaviors), the dataset to use, as well as other parameters of interest. In this mode, training always starts from an untrained model and proceeds until the specified number of rounds.

3.6.2 Deploying an ML Model

Models can also be retrieved and deployed programmatically using the dedicated API endpoint:

GET /api/robust/model/{scenario_name}

This endpoint retrieves models generated during training in a specific scenario. By default, it returns to the latest global model, but it also supports flexible filtering options:

- **participant (int, optional)**: it retrieves a model produced by a specific participant node (e.g., *participant_2_round_10_model.pth*).
- **metric (string, optional)**: it retrieves the best model according to a performance metric (F1 Score, accuracy, precision, recall, loss).
- **round (int, optional)**: it retrieves the model checkpoint corresponding to a specific round of training.

Three examples of requests for executing different functionalities are shown below.

- Latest global model of scenario *demo_scenario*:

  curl -X GET "http://localhost:8000/api/robust/model/demo_scenario"

- Model from participant 1 at round 5:

  curl -X GET "http://localhost:8000/api/robust/model/demo_scenario?participant=1&round=5"

- Best model according to accuracy:

  curl -X GET "http://localhost:8000/api/robust/model/demo_scenario?metric=accuracy"

Deployment with TorchServe

Trained models exported as *.pth* can be deployed as scalable inference services using TorchServe [TS25], a flexible, scalable server for deploying PyTorch models easily in production environments. This process involves:

1. **Preparing the artefacts**. This initial step involves preparing the model and its associated logic for serving. There are two primary approaches:
   a. **Eager Mode with a Custom Handler:** In this approach, the standard PyTorch model saved as a .pth file is used directly. However, it requires a custom handler, which is a Python script (.py) that acts as the logic wrapper for the model. The handler defines three critical functions: initialize (to load the model into memory), preprocess (to transform raw input data, like a JSON payload or an image, into the tensor format the model expects), and postprocess (to convert the model's output tensor back into a human-readable format, like JSON). Preparing the handler involves writing this script, ensuring it correctly loads the .pth file and handles the data formats expected by the model and the client.
   b. **TorchScript Export:** Alternatively, the model can be exported to the TorchScript format (.pt). This process compiles the PyTorch model into a self-contained, JIT-compiled graph. The main advantage is that it removes the dependency on the original Python model

definition code, which can improve performance and portability. Even when using TorchScript, a custom handler is often still necessary for the pre- and post-processing steps.

2. **Packaging the model**. Once the artifacts are ready, they are bundled into a single, deployable unit called a Model Archive (.mar). This file is created using the torch-model-archiver command-line tool.
    a. **Serialized Model File:** The model weights, either as a .pth (for eager mode) or .pt (for TorchScript).
    b. **Handler Script:** The custom Python handler script (e.g., handler.py) created in the previous step.
    c. **Model Definition File (if required):** For models in eager mode, the Python file containing the model's class definition is required so that TorchServe can instantiate the model architecture before loading the weights.
    d. **Auxiliary Files:** Any additional files required by the handler, such as an index_to_name.json file for mapping class indices to human-readable labels, or other configuration files.
    e. **Dependency Manifest:** A requirements.txt file can be included to specify any Python libraries (e.g., numpy, Pillow) needed by the handler. TorchServe will automatically install these dependencies in its environment, ensuring portability.
3. **Configuring TorchServe.** TorchServe itself can be configured via a config.properties file. This allows for defining how models are loaded, managed, and exposed. Key configurations include setting the inference address, the number of workers per model for parallel processing, batch size, and the initial models to be loaded on startup.
4. **Containerization for Deployment.** The final step involves creating a self-contained and portable service using containerization technologies like Docker. A Dockerfile is created to assemble a Docker image that embeds the TorchServe instance and the model to be deployed. The image typically includes:
    a. The official TorchServe base Docker image.
    b. The packaged Model Archive (.mar) file, copied into a dedicated model-store directory within the image.
    c. The custom config.properties file defines the server's behavior.

The ENTRYPOINT of the Docker image is configured to start the torchserve process. Upon container startup, the TorchServe instance inside the container automatically scans the model-store directory, loads the model specified in its configuration, and exposes its REST/gRPC inference endpoints. Thus, the container itself becomes the deployable unit, containing both the serving engine and the model, ready to be served immediately. This workflow ensures that models can be seamlessly transitioned from experimental artifacts to reproducible, portable, and scalable services for local inference, benchmarking, or deployment in containerized environments such as Kubernetes.

3.7 Integration of future customizations

A key strength of the DFL Framework lies in its modular and extensible architecture, which has been deliberately designed to facilitate the integration of new functionalities with minimal overhead. This flexibility ensures that the framework can evolve alongside emerging research challenges, enabling the consortium and external contributors to enrich its capabilities continuously. To do this, the integration process is streamlined by standardised interfaces and containerisation, ensuring compatibility and reproducibility across extensions.

3.7.1 Datasets/Models

The DFL Framework allows the incorporating of new datasets and custom model architectures in a seamless manner.

- **Datasets**: Datasets can be added to broaden the applicability of the framework. Both IID and non-IID partitioning strategies are supported, enabling experiments that replicate realistic data heterogeneity across Nodes.
- **Models**: Any external entity can integrate different neural network architectures by plugging them into the training pipeline with minimal modifications, such as ResNet-18, ViT, or LSTM.

3.7.2 Aggregators

The aggregation process is central to federated learning, and the DFL Framework provides a modular interface for incorporating new aggregation algorithms. While Federated Averaging (FedAvg) is supported as the default baseline, any entity can easily replace or complement it with advanced strategies, such as:

- **Robust aggregation methods** for resilience against adversarial updates.
- **Privacy-preserving techniques** (e.g., secure multiparty computation, differential privacy-enhanced aggregation).
- **Adaptive strategies** that dynamically weight contributions based on trust, reliability, or energy efficiency.

By supporting such flexibility, the framework enables experimentation with state-of-the-art aggregation approaches and facilitates comparative evaluation across diverse robustness and privacy criteria.

3.7.3 Adversarial Attacks

Security evaluation is a fundamental objective of the DFL Framework, and its architecture supports the integration of novel adversarial strategies for systematic testing [MSB+24].

Examples include:

- **Data poisoning**: Subtle or large-scale corruption of training data (e.g., using label flipping) at malicious nodes.
- **Model poisoning**: Manipulation of the local training process or the generated model updates to deliberately degrade the performance of the global model or bias its behaviour.

The inclusion of these strategies allows for simulating realistic adversarial environments to evaluate the robustness of different defence mechanisms. Since these attacks can be parameterized (e.g., intensity, frequency, percentage of compromised Nodes), the DFL Framework supports a wide range of adversarial scenarios, from stealthy to overt.

4 Trustworthiness module

4.1 Introduction and purpose of the module

Trustworthiness has become a foundational concern in modern communication networks, particularly as AI systems are increasingly embedded. Although AI can significantly enhance functionality and decision-making, it may raise issues around reliability, transparency, accountability, privacy and ethical alignment. The “Trustworthy AI module” is devoted to increase trustworthiness of AI by making AI more robust against adversarial attacks and enhancing privacy. These two aspects are critical to ensure that AI services operate reliably under adversarial conditions and that sensitive information used during model training remains secure. To demonstrate how this module interacts with other modules across different layers and work packages, the solutions and corresponding flows are provided in the following subsections.

We will start with the vanilla FL service and relate it with a telco-specific use case scenario which utilize AI-driven security solutions. Then, we will be investigating the same use case scenario with an additional need for privacy-enhanced FL service. In this part, we will elaborate how privacy enhancing mechanisms (i.e., homomorphic encryption) can be applied to protect the model updates that are shared during learning phase between the clients and the FL server. Then, we will mention about Robust FL Services where we extend the use case scenario by incorporating robustness mechanisms that defend the collaborative learning process against adversarial manipulation in the learning phase. Finally, we will be discussing the robustness aspect and mention about how we can introduce robustness to AI-driven solutions and how we can protect them from adversarial attack scenarios. Enhancing robustness is a key enabler of trust in FL services, as adversarial attacks such as poisoning can severely degrade model accuracy, distort decision outcomes, and even leak private information through manipulated updates. When clients or stakeholders cannot rely on the integrity of the learned model, both the utility and the perceived fairness of the service are compromised. Robust aggregation algorithms (e.g., Krum, Trimmed Mean, FoolsGold) strengthen the model against such risks by reducing the impact of corrupted updates, while poisoning detection mechanisms can proactively flag suspicious contributions. Together, these techniques not only preserve the reliability and utility of the global model but

also reinforce confidence that the system remains secure and privacy-preserving in adversarial environments, thereby achieving true trustworthiness in AI-driven telco applications.

4.2 Trustworthiness Module Design

One of the modules within the Trustworthy AI service layer is the Enhanced Federated Learning (FL) Services module.

4.2.1 Vanilla FL Service

In Figure 4-1, a flow diagram illustrates the process by which a vanilla FL service is triggered. It also visualizes how this request propagates through the various layers of the system and highlights the interactions between the involved Work Packages (WPs). As it is illustrated in Figure 4-1, an external consumer/application requests RF-fingerprinting based security (with different motivations such as identification, authentication, authorization) service from the operator for certain location area, and forwards the request via the exposure framework. We assumed that Radio devices (i.e., gNBs) register themselves as potential "clients" to the FL repository beforehand (each client may store supported features related to model training, such as current traffic load, if it has a Graphics Processing Unit (GPU), which types of models it can train, model architecture, etc.).

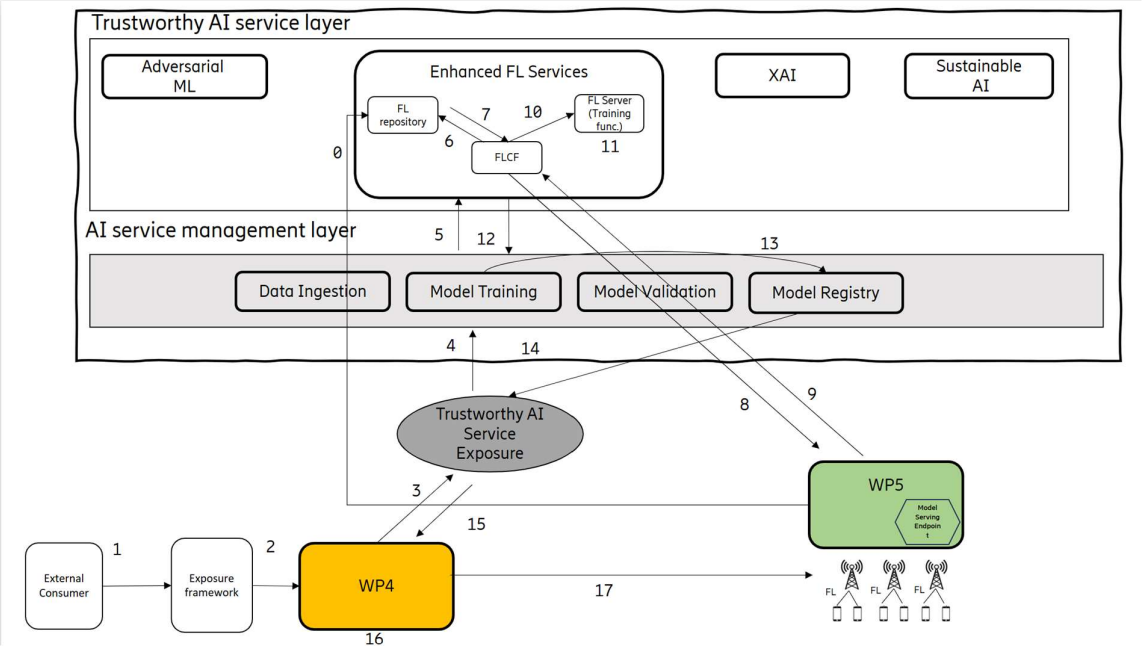

Figure 4-1 Vanilla FL service flow

The description of each of the steps in the flow is as follows:

0 - (independently) Radio devices (i.e., gNB's) register themselves as a potential "client" role to the FL repository beforehand (each client may store supported features related to model training, such as current traffic load, if it has a Graphics Processing Unit (GPU), which types of models it can train, model architecture, or may also the privacy operations).

1 - External consumer/application requests privacy protected RF-fingerprinting based security (identification/authentication/authorization) service from the network provider for certain location area and forwards the request via the exposure framework.

2 - The exposure framework receives this security service request and forwards it to the zero-touch security management layer.

3 - ZSM understands that for this specific security service, privacy enhanced FL-based AI model would be a better option. It sends an FL model request via Trustworthy AI service exposure (target location is also added as info).

4 - Trustworthy AI service exposure forwards this privacy enhanced FL model request to the AI service management layer.

5 - The AI service management layer forwards this request to the Enhanced FL services component in the Trustworthy AI service layer.
6 - Federated Learning Coordination Function (FLCF) in Enhanced FL services component sends a client discovery to FL repository.
7 - FL repository notifies the list of suitable clients according to their supported features and the target location.
8 - FLCF sends FL participation requests to the list of provided clients.
9 - Acknowledgement of the participation by the clients are sent back to FLCF.
10 - FLCF sends a message to FL Server to start the FL process together with the participated clients.
11 - FL Training takes place and clients are triggered to start training.
12 - Trained model is sent to AI service management layer.
13 - Trained model is registered (saved) to Global Model Repository (GMR).
14 - Trained model is forwarded to Trustworthy AI service exposure.
15 - Trained model is sent back to Zero-touch security management layer.
16 - Zero-touch security management layer bundles this model with the appropriate security application to be deployed to target environment (i.e., radio units).
17 - Zero-touch security management layer deploys the final security application to the target environment.

The above-mentioned flows are purely theoretical that will not be implemented and are only an example of how vanilla FL service could be used.

4.2.2 Privacy-enhanced FL Services

In this subsection, we consider a scenario in which a server may behave maliciously and attempt to extract sensitive information from individual clients by analysing the model updates it receives. In such cases, enhanced privacy protection mechanisms are required during the collaborative learning process to safeguard the potentially sensitive data of participating clients. We extend the above use-case scenario by introducing additional steps to coordinate the application of privacy enhancing mechanism on client model updates in the collaborative learning process. Figure 4-2 is the flow diagram for such a solution, along with the details of each step (additional steps are provided in italicized text). The new steps in the flow are added leveraging the privacy protection framework that is defined in our recently accepted paper [KKF+25]. The Privacy Management Function (PMF) could be an entity inside the FLCF that responsible for selecting the privacy enhancing mechanisms, and generating and distributing of related key pairs and parameters.

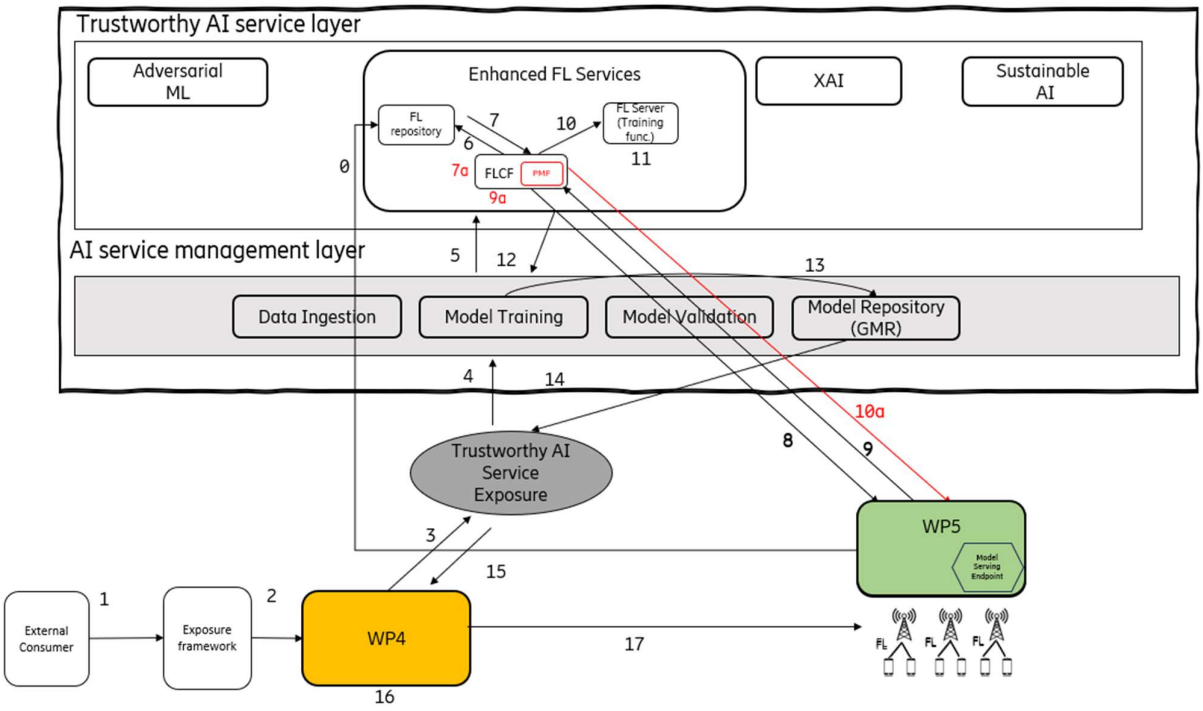

Figure 4-2 Privacy-enhanced FL service flow

0 - (independently) we assumed that Radio devices (i.e., gNB's) register themselves as a potential "client" role to the FL repository beforehand (each client may store supported features related to model training, such as current traffic load, if it has a Graphics Processing Unit (GPU) , which types of models it can train, model architecture, or it may also be the supported privacy enhancing mechanisms).

1 - External consumer/application requests RF fingerprinting based security (identification/authentication/authorization) service from the operator for certain location area and forward the request via the exposure framework.

2 - The Exposure framework receives this security service request and forwards it to the zero-touch security management layer.

3 - The Zero-touch security management layer understands that for this specific security service, FL based AI model is needed. It sends an FL model request via Trustworthy AI service exposure (target location is also added as info).

4 - Trustworthy AI service exposure forwards this FL model request to AI service management layer.

5 - The AI service management layer forwards this request to Enhanced FL services component in Trustworthy AI service layer.

6 - The Federated Learning Coordination Function (FLCF) in Enhanced FL services component sends a client discovery to FL repository.

7 - FL repository notifies the list of suitable clients according to their supported features including supported privacy enhancing mechanisms and the target location.

   a. The FLCF identifies a list of clients and the privacy enhancing mechanism to be used by taking clients' supported features including supported privacy enhancing mechanisms into account.

8 - FLCF sends FL participation requests to the list of identified clients (radio units).

9 - Acknowledgement of the participation by the clients are sent back to FLCF.

   a. The FLCF obtains required parameters for the selected privacy enhancing mechanism from PMF (Privacy Management Function).

10 -FLCF sends a message to FL Server to start the FL process together with the participated clients and the privacy enhancing mechanism parameters.

   a. The FLCF sends the privacy enhancing mechanism parameters to the clients who accepted to be involved in the FL.

11 -FL Training takes place and clients are triggered to start training. FL training is executed in a privacy enhanced way by using the selected privacy enhancing mechanism.

12 -Trained model is sent to AI service management layer.

13 -Trained model is registered(saved) to Global Model Repository.
14 -Trained model is forwarded to Trustworthy AI service exposure.
15 -Trained model is sent back to Zero-touch security management layer.
16 -Zero-touch security management layer bundles this model with the appropriate security application to be deployed to target environment (radio units).
17 -Zero-touch security management layer deploys the final security application to the target environment.

The above-mentioned flows are purely theoretical that will not be implemented and are only an example of how privacy-enhanced FL service could be used.

4.2.3 Robust FL Services

We extend the use-case scenario by incorporating robustness mechanisms that defend the collaborative learning process against adversarial manipulation. These mechanisms focus on identifying and mitigating the impact of poisoned updates before they are aggregated into the global model, thereby preserving both accuracy and trust in the service outcome. The flow in Figure 4-3 illustrates how robustness checks are embedded into the FL pipeline, with specific steps dedicated to adversarial testing and poisoning detection integrated alongside the standard training and validation stages.

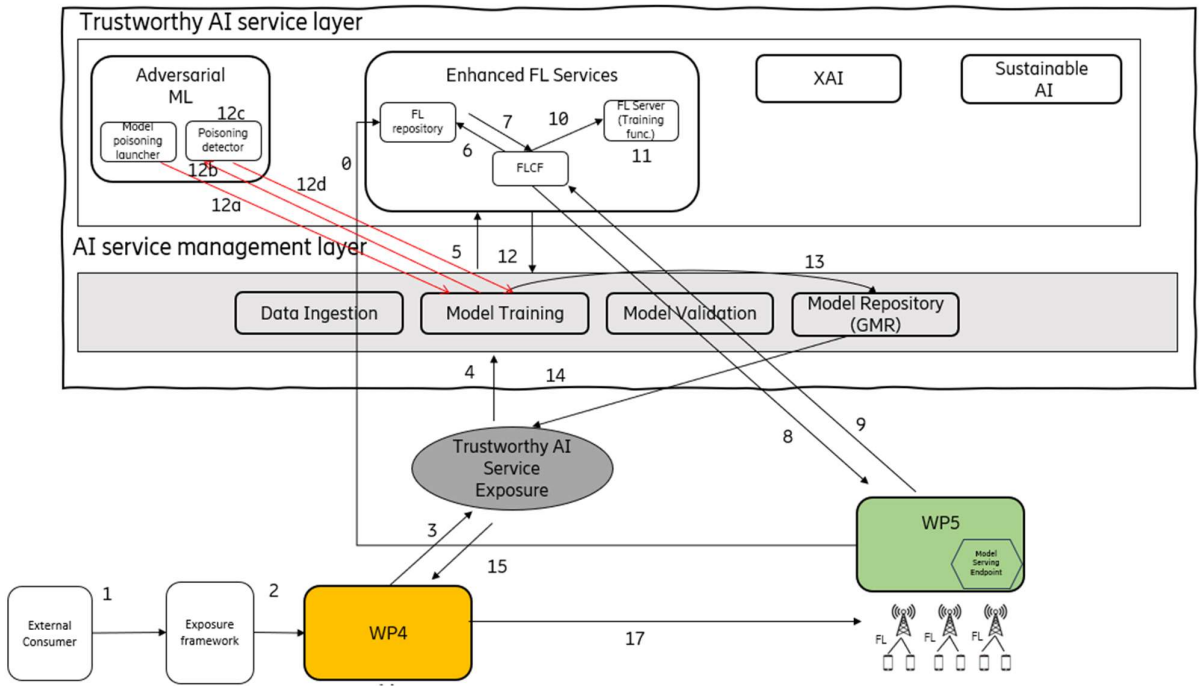

Figure 4-3 Robust FL service flow

0 - (independently) we assumed that Radio devices (i.e., gNBs) register themselves as a potential "client" role to the FL repository beforehand (each client may store supported features related to model training, such as current traffic load, if it has a Graphics Processing Unit (GPU) , which types of models it can train, model architecture, or may also the privacy operations).
1 - External consumer/application requests RF-fingerprinting based security (identification/authentication/authorization) service from the operator for certain location area and forward the request via exposure framework.
2 - The Exposure framework receives this security service request and forwards it to zero-touch security management layer.
3 - The Zero-touch security management layer understands that for this specific security service, FL based AI model is needed. It sends an FL model request via Trustworthy AI service exposure (target location is also added as info).
4 - The Trustworthy AI service exposure forwards this FL model request to AI service management layer.
5 - The AI service management layer forwards this request to Enhanced FL services component in Trustworthy AI service layer.

6 - Federated learning coordination function (FLCF) in Enhanced FL services component sends a client discovery to FL repository.
7 - FL repository notifies the list of suitable clients according to their supported features and the target location.
8 - FLCF sends FL participation request to the list of provided clients (radio units).
9 - Acknowledgement of the participation by the clients are sent back to FLCF.
10 -FLCF sends a message to FL Server to start the FL process together with the participated clients.
11 -FL Training takes place, and clients are triggered to start training.
12 -The trained model is sent to AI service management layer.
    a. Poisoned models are (optionally) added with model poisoning for robustness testing.
    b. Multiple trained models are evaluated for poisoning by the poisoning detection in the adversarial ML.
    c. Suspicious models are flagged as poisoned with the detection outcomes.
    d. Remaining models are forwarded to the model registry.
13 -The trained model is registered(saved) to Global Model Repository (GMR).
14 -The trained model is forwarded to Trustworthy AI service exposure.
15 -The trained model is sent back to Zero-touch security management layer.
16 -ZSM bundles this model with the appropriate security application to be deployed to target environment (radio units).
17 -ZSM deploys the final security application to the target environment.

The above-mentioned flows are purely theoretical that will not be implemented and are only an example of how robust FL service could be used.

4.2.4 Robust AI Services

We focus on the **robustness** aspect and propose a solution for introducing robustness to AI-driven solutions and protecting them from adversarial attacks. We consider a separate Adversarial ML module within Trustworthy AI services layer which is working in a coordinated manner with the basic MLOps components within the AI Services Management Layer such as data ingestion, model training, model validation and model registry. The flow diagram in Figure 4-4 illustrates the robust training process which includes adversarial sample generation, adversarial training, robustness evaluation, and certification before model registration. The following provides explanations of the components depicted in the flow diagram in Figure 4-4.

- Data Ingestion (DI): supplies curated training data with lineage/quality tags.
- Model Training (TR): trains the baseline model but now augmented with adversarial examples.
- Adversarial ML Module (AML, in Trustworthy AI Layer):
    - Generates adversarial examples during training.
    - Provides robustness certification during validation.
- Model Validation (VA): checks both accuracy and robustness metrics (e.g., certified robustness accuracy, OOD detection score).
    - RA (ε) metric shows Robust Accuracy at ε. It is simply the percentage of inputs on which a model gives the correct output even when those inputs are perturbed by an adversary within a bounded distortion level (ε). It can provide a certified robustness measure against small adversarial perturbations.
    - OOD AUC stands for the Out-of-Distribution Detection Area Under Curve. It is considered as a measure of how well a model (or detector) can distinguish in-distribution inputs (the data it was trained on) from out-of-distribution (OOD) inputs (unseen or anomalous data).
    - And finally Backdoor Test / ASR (Attack Success Rate under Backdoor) measures a model's vulnerability to backdoor or trojan attacks (where an attacker poisons training data so that a trigger pattern in input causes malicious misclassification).
- Model Registry (RG): stores the final model together with robustness certificates and metadata for deployment.

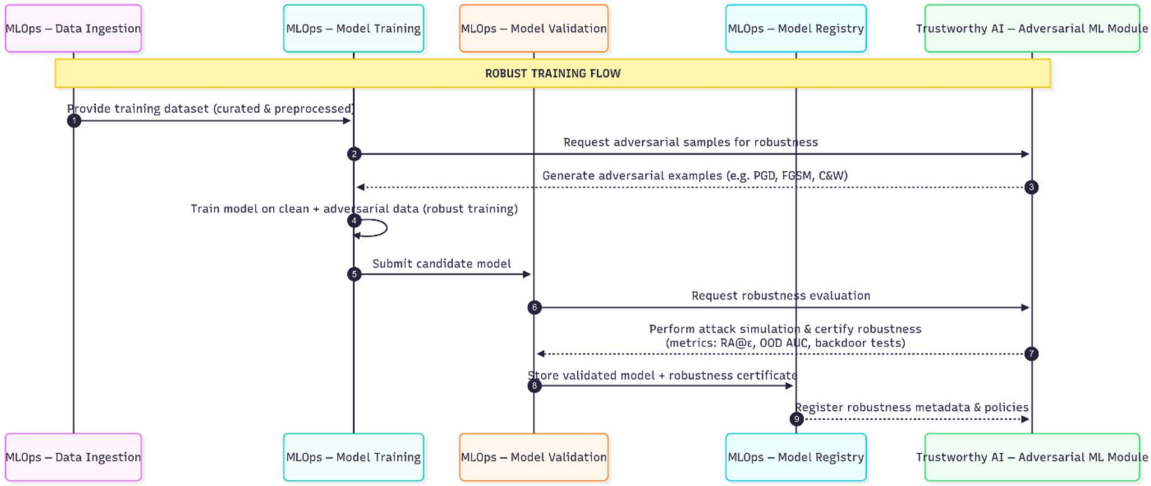

Figure 4-4 Robust AI service flow

5 Sustainability module

5.1 Introduction and purpose of the module

Sustainability is a fundamental concern of modern communication networks, especially for those that employ massively AI in the loop, either for the optimization of other processes (e.g., industry and infotainment), or of the network itself. Albeit AI improves the efficiency in a variety of contexts, it comes with an additional computation overhead that often leads to a significant ecologic footprint.

The “Sustainable AI module”, introduced in

Figure 2-1, is devoted to reducing such ecologic footprint by making AI more energy efficient and scalable. Specifically, this is achieved by:

- Optimizing AI training algorithms for the edge and cloud networks via the exploitation of specific energy- and semantics-aware contexts, and
- Designing efficient ML models for deployment on networked and low power devices.

5.2 Sustainable AI module design

Figure 5-1**Error! Reference source not found.** shows the diagram of the Sustainable AI module, which is the result of grouping and integrating three individual components that were previously referred to in D3.2 and in D6.1. The module includes blocks that:

1. Analyze the arrival of requests through the prototype’s API from outer ROBUST-6G layers and, possibly, other external processes defining sustainability requirements and providing any relevant input context;
2. Offer sustainable-by-design ML model (optional, depending on the requirements), service offered by CUPD02 (D3.2);
3. Select a sustainable and scalable distributed training algorithm, including i) aggregation method (CUPD01, D3.2), and ii) context-aware client scheduling (CLIU01, D3.2, optional, depending on the requirements);
4. The evaluation of the model dumped in the GMR by measuring the energy consumption of training related to the model performance in terms of accuracy/F1-score, semantics-based metrics, energy consumption at inference, etc.

It is important to highlight that some of the services offered by the Sustainable AI module are integrated with the DFL Framework (see Section 3), specifically, point 3 of the previous list. However, these services can be also used when the model training is not performed through the DFL Framework but, e.g., through standard server-based FL. Also, the sustainable AI module can still offer the services at points 2 and 4 of the previous list when model training is performed in a centralized way, as model selection is often agnostic to the training

method used, and the interaction with the GMR of the module is independent of the training method. In the following, we delve into the details of each of the points above listed.

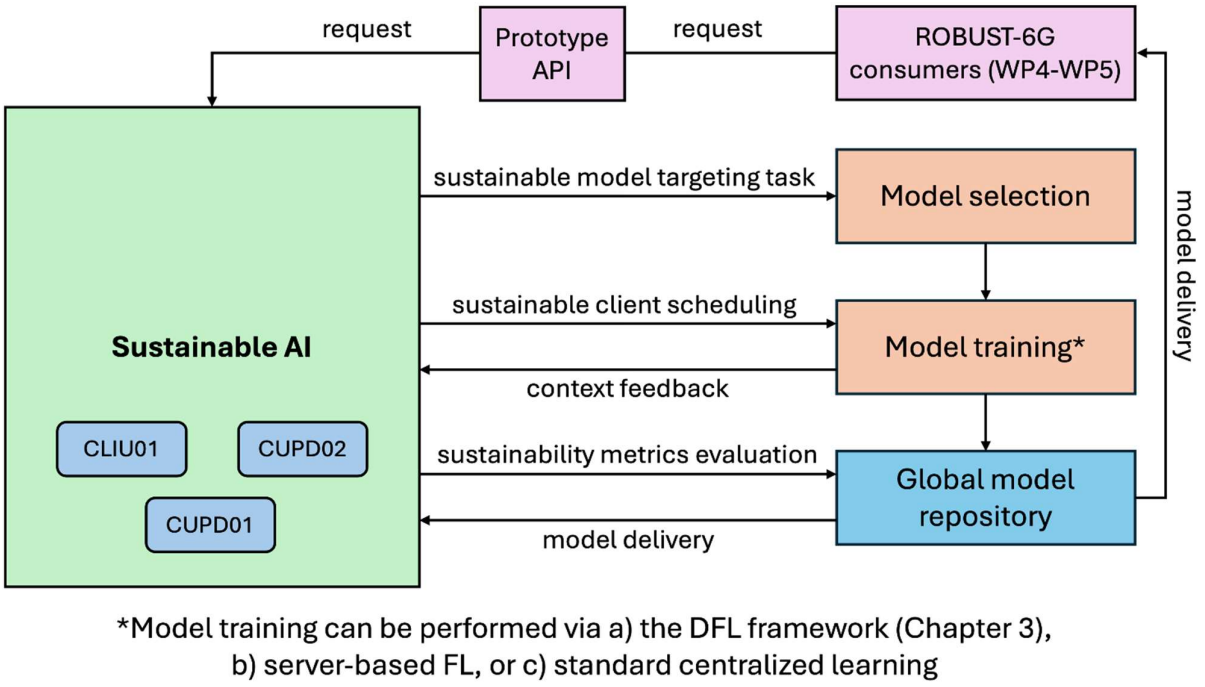

Figure 5-1 Diagram of the Sustainable AI module and functionality flows

5.2.1 Specification of requirements from outer ROBUST-6G network layers

Other ROBUST-6G layers, i.e., the ZSM and the network (NET) layers can retrieve AI models they asked for in the GMR (through the prototype's API, see Section 2). For this, they must provide the *task* the model has to solve and the training *dataset* (and, possibly the dataset *split*).

However, they can also specify whether the model is the result of training in a federated way, server-based or serverless, or if it is trained in a centralized fashion. This reflects different scenarios and applications that can be relevant in networks. Together with the *training mode*, consumers can also specify training contexts having notable implications in energy efficiency, e.g., the use of *renewable and intermittent sources*, or *bounds to the computational power* of training devices. Along with these relevant environment contexts and device characteristics, *target sustainability requirements* can also be requested, e.g., that the model consumes at most a certain amount of energy at inference because it must run on low power connected devices.

5.2.2 Sustainable models

The Sustainable AI module offers models that are energy efficient by design when running on dedicated neuromorphic hardware [RBG+22]. Specifically, UNIPD worked on *spiking neural networks (SNNs)* and their improvement in terms of accuracy in [APM+25] and [ARC25], as well as on a novel training method specifically tailored for these neural networks [PBM+25]. These functionalities are provided by the component that was previously labeled CUPD02. A study from Intel shows that the energy-delay-product (EDP) of SNNs, i.e., a metric measuring a tradeoff between the inference latency and energy consumption, is three orders of magnitude lower than the one of a traditional NN [RBG+22]. Notably, SNNs are especially thought for processing time-series and signals in real-time, which is relevant to ROBUST-6G for security monitoring both at the ZSM and at the NET.

Other than that, the Sustainable AI module offers post-training optimization of models for better memory and computational footprint, using *quantization* and *pruning* context-agnostic methods. If this is done iteratively during training, the procedure lowers also the training energy consumption.

5.2.3 Sustainable client scheduling

The Sustainable Client Scheduling component (previously labeled CLIU01) implements mechanisms that determine client participation in FL rounds based on energy availability, network conditions, and computational capacity. Each component in this module contributes to the overall objective by addressing a distinct dimension of sustainability, ranging from client-level scheduling to model-level computational efficiency. The module can be consumed either as a standalone Python library and scripts or as an integrated service within the DFL framework. In the first consumption type, users specify client conditions such as battery traces, communication parameters, and model types to obtain scheduling decisions as output. On the other hand, if the module is integrated into the framework, scheduling can be invoked automatically through configuration files, which define client resource profiles and training policies.

Energy-aware scheduling enables clients to time their local training and transmissions according to their battery states, avoiding unnecessary updates and stabilizing participation across rounds [JP25]. Decentralized scheduling mechanisms based on peer-to-peer consensus allow clients to coordinate updates without relying on a central server, distributing communication load and adapting scheduling decisions to network conditions. Finally, spiking neural network–based training supports task allocation that accounts for computational efficiency, where clients with limited capacity can be scheduled to run lightweight, event-driven updates rather than conventional dense training.

The sustainable scheduling mechanisms have been validated through simulation-based test cases. In particular, the energy-aware scheduling scheme was evaluated with client energy traces, showing improved participation stability and the tradeoff between accuracy and energy consumption. The test cases confirm that the scheduling decisions can be reproduced and assessed by other components [JP25, BDP+25].

The scheduling mechanisms provide architecture-agnostic features to the system in that they are operatable within the DFL Framework but also compatible with standard server-based FL, even in scenarios with high user mobility [BDP+25]. This integration flexibility allows to enable the sustainability across different learning paradigms depending on the deployment requirements. This module, thus, defines a sustainable framework in which who participates, when they participate, and under what computational constraints are all considered.

5.2.4 Sustainable and scalable aggregation methods

Scalable aggregation methods for DFL and server-based FL are important to improve the sustainability of networked training procedures as they make a better use of network resources by i) lowering the number of communication rounds, and ii) increasing the pool of devices aggregating information, hence improving the model knowledge and dataset variety.

Through the component previously labeled CUPD01, the Sustainable AI module uses established optimization algorithms (the alternating direction method of multipliers (ADMM)) to reach consensus through message passing. The procedure extends naturally for any kind of network topology, whether it be the star of server-based FL or a more general mesh for DFL.

5.3 Models and metrics stored in the Global Model Repository

The following models and metrics produced by the Sustainable AI module are stored in the GMR. Depending on the integration scenario, these models may originate from the DFL Framework, from standard server-based FL, or from centralized learning.

Models

- Battery-aware models: models trained while leveraging the battery-aware cyclic scheduling, or using intermittent renewable energy sources (e.g., solar).
- Decentralized ADMM models: trained with the decentralized consensus for scalability.
- SNN energy-efficient models: trained with event-driven updates/datasets.

Metrics

- **Energy-aware metrics**
  - Per-client and system-wide energy consumption;
  - Ratio of computation to communication energy.
- **Learning performance metrics**

  - Global accuracy;
  - F1-score;
  - Stability across heterogeneous clients.
- **Scheduling metrics**
  - Stability of client selection under resource constraints;
  - Client availability traces.
- **Model inference energy metrics**
  - Energy consumption at inference;
  - Number of FLOAT operations for a forward pass;
  - If SNN: spiking activity rate (SAR).

6 Explainability module

6.1 Introduction

The transparency and accountability of AI systems, two of the various aspects of trustworthy AI, represent a critical requirement within communication networks, especially in environments where AI/ML algorithms are extensively integrated across operational workflows. While these systems show remarkable performance improvements across many application domains, they simultaneously introduce significant opacity that undermines user confidence and the demand for better explanation processes of these systems. Explainable AI addresses these challenges by making AI systems more transparent, interpretable, and robust, thereby enhancing trust, enabling better decision-making processes, and ensuring reliable performance in complex operational environments. Otherwise stated, AI systems not only need to make predictions, but also know when they are uncertain and explain why.

This module addresses the explainability gap by pursuing two complementary strategies. Each strategy consists of two services – methodologies:

- The first strategy focuses on three different components each providing a unique aspect to create explainable services. The first service focuses on developing model agnostic confidence metrics leveraging conformal prediction. Conformal prediction is a powerful framework that enables ML models to generate predictions with guaranteed error rates. Unlike traditional confidence scores, conformal predictors offer rigorous and statistically valid measures of uncertainty. These guarantees hold regardless of the underlying model or data distribution, making conformal prediction exceptionally robust and broadly applicable. The first proposed metric is termed Conformal Prediction Confidence Factor (CPCF) and quantifies the model's confidence on learned tasks and offers an explainable metric that reveals how the introduction of new learning tasks or data affects the model's ability to accurately predict on historical tasks or data. This method improves prediction reliability, providing dynamic confidence intervals in order to provide an explanation for the quantification of the model's output confidence. The second method/service incorporates a model-specific confidence estimation approach using Variational Autoencoders (VAEs) that leverages latent space representations to assess prediction reliability. This method measures the proximity of the latent representation and provides a confidence metric about the new predictions. The third method/service extends a version of the SHIELD (Selective Hidden Input Evaluation for Learning Dynamics) regularisation technique augmented with Bayesian neural network capabilities by employing the IVON (Improved Variational Online Newton) optimizer. This approach applies SHIELD techniques to evaluate selective hidden input evaluation, ensuring transparent decision-making processes while enhancing overall model performance through regularization-derived explainability metrics. This integration enables the service to provide probabilistic explanations that help network operators understand not just which features are important, but also the confidence levels associated with these explanations.

- The second strategy offers two distinct services that can be migrated together by leveraging XAI for model improvement and post-hoc analysis. The first service provides XAI-driven model optimization, refining a given security model by identifying and retaining only its most impactful input features. The second service delivers explainability as a metric, generating detailed incident reports that decipher the reasoning behind a model's predictions in a human-understandable format. While these services

can be used in tandem, generating reports for an optimized model, they can also be engaged independently.

Together, these pillars establish a holistic framework for XAI, ensuring that AI-driven security functions are not only accurate but also transparent, trustworthy, and explainable. In addition, the pre-trained models stored in the Global Model Repository (GMR), they serve as companion outputs. Specifically, the services described above contribute two key artefacts to the GMR: a high-performing, efficient model and its corresponding explainability and confidence metrics.

The Explainable AI module addresses this explainability gap through the development of transparent and accountable machine learning frameworks. The module's core objectives contain:

- Enhancing algorithmic transparency in distributed computing environments through the implementation of interpretability-first design principles.
- Developing explainable ML architectures suitable for deployment across different networked infrastructures.
- Optimizing security models by employing XAI-driven feature selection to reduce data dimensionality, remove noise, and enhance computational efficiency.
- Providing post-hoc, model-agnostic explanations for security incidents by analyzing the features that drive a model's predictions.
- Producing standardized, machine-readable, and human-understandable metrics (e.g., JSON-based incident reports) for the Global Model Repository to ensure interoperability.
- Improving network operator trust and enabling more informed, rapid decision-making by translating complex model behaviour into actionable insights.
- Establishing a foundation for fair and accountable AI by making the logic of security models auditable and transparent.

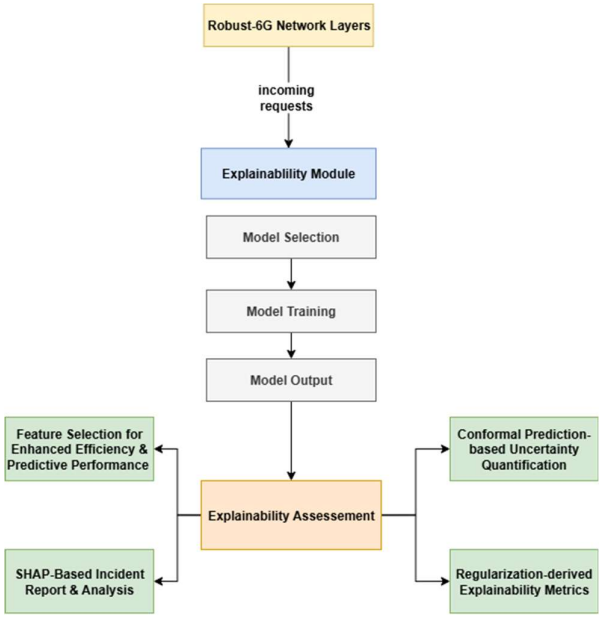

Figure 6-1 Diagram of the Explainability module and provided services

6.2 Explainability Module Offered Services

The Explainability module, as shown in Figure 6-1, will include four main services. The description of this module is as follow:

1. The arrival of incoming requests processed through the ROBUST-6G Network Layers, defining explainability requirements and providing the operational context for subsequent model deployment and assessment procedures.

2. The execution of model selection processes within the XAI framework, determining optimal architectures based on interpretability constraints and performance requirements.

3. The implementation of model training protocols enhanced with transparency-preserving algorithms and SHIELD-based regularization techniques, incorporating context-aware training optimization to ensure explainability is embedded throughout the learning process.

4. The generation of model outputs serving as the primary inference results, which subsequently feed into the explainability assessment pipeline for comprehensive interpretability evaluation.

5. The deployment of the **Explainability Assessment** component, which orchestrates a multi-faceted evaluation by coordinating the following services and analyses:
    a. **Conformal Prediction-based Uncertainty Quantification**: Implements the CPCF methodology to assess prediction reliability, providing dynamic confidence intervals to provide explanations for the quantification of the model's output confidence.
    b. **Latent Space-based Confidence Estimation:** Implements a VAE methodology to assess prediction confidence through the proximity of the unknown representation and the training points on the Latent Space regarding the Mahalanobis distance.
    c. **Regularization-derived Explainability Metrics**: Applies SHIELD techniques to evaluate selective hidden input evaluation to ensure transparent decision-making processes while enhancing overall model performance.
    d. **Feature Selection for Enhanced Efficiency Predictive Performance**: Employs XAI-based techniques to identify and select the most critical input features, which improves model efficiency and predictive power while providing a clear understanding of the data driving the outcomes through an incident report.
    e. **SHAP-Based Incident Report & Analysis**: Utilizes SHapley Additive exPlanations (SHAP) to generate detailed reports that explain individual predictions, offering precise, quantifiable insights into how each feature contributed to a specific result.

6.3 Design and Implementation of Explainability Services

6.3.1 XAI-driven Model Optimization and Refinement

High-dimensional data, common in network traffic analysis, can introduce noise and computational overhead, potentially degrading the performance of security models, especially for rare attack types. To address this, the module incorporates an advanced, XAI-driven feature selection framework called **SHAPRefine**. This process moves beyond using explainability as a post-hoc analysis tool and embeds it directly into the model optimization pipeline. The framework is fundamentally model-agnostic, capable of working with any baseline model and a wide array of input data. While theoretically applicable to any high-dimensional, multiclass classification problem, its initial implementation is tailored for Intrusion Detection Systems (IDS), where the input consists of network flows, features, or logs.

The methodology leverages SHAP (SHapley Additive exPlanations) values to systematically evaluate the importance of each feature. Aggregation of these values is done based on class frequency, distribution, and contribution to the model's predictions. Through a performance-driven refinement strategy, features are iteratively assessed and only retained when increasing the cross-validation metrics. By focusing on features with the highest impact, this approach produces lightweight, high-performing models tailored for real-time deployment in demanding network environments.

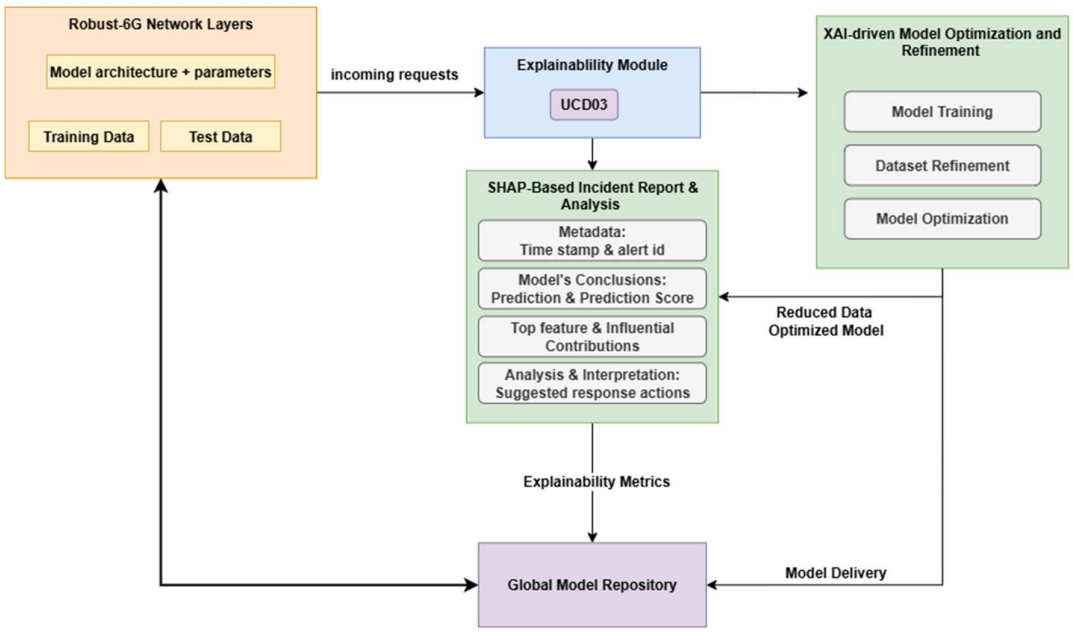

Figure 6-2 Diagram of the UCD03 Explainability component with outputs to GMR

The primary output of this service is a refined and optimized model, which is not only more efficient and compact due to reduced dimensionality but also exhibits higher predictive accuracy and robustness. This resulting artefact—a lightweight, high-performing model trained on the reduced feature set—is designed to be stored in the GMR. From there, it can be accessed by other partners and system components, ensuring that the entire architecture benefits from a more efficient and robust security model.

6.3.1.1 Implementation details

The framework consists of four main stages:

1. **Baseline model training.** The input to the service is training data provided by other ROBUST-6G members. This data should ideally be a labeled classification dataset with multiple features. Since the feature selection method is model-agnostic, partners are free to choose their preferred baseline model. By default, SHAPRefine employs an **XGBoost classifier** as a robust and well-tested option.
2. **SHAP value computation.** Once the baseline model is trained, **Shapley values** are computed to quantify the contribution of each feature across all attack classes. These values capture both global and class-specific importance, forming the foundation for refinement.
3. **Aggregation and weighting.** To ensure fair representation of minority attacks, class-wise SHAP scores are normalized and weighted according to class frequency. This step prevents dominant classes from overshadowing rare but critical attack types.
4. **Iterative refinement.** Features are incrementally added based on their aggregated importance scores, with inclusion permitted only if they improve cross-validation metrics. This process ensures that the final feature set is compact, synergistic, and performance-driven rather than arbitrarily reduced.

This component has been rigorously validated using benchmark IDS datasets to represent different network enviromemnts and the durability of this component:

- **Dimensionality reduction** of up to *90%*,
- **Training time and energy consumption reductions** exceeding *90%*,
- **Detection accuracy improvements**, with minority-class F1-scores boosted by up to *95%*.

6.3.1.2 Integration with ROBUST-6G

Within the ROBUST-6G system, SHAPRefine is positioned as an **optional optimization service** that enhances the IDS pipeline. It consumes high-dimensional training data or baseline models produced by other WP2 components and refines them into lightweight, high-accuracy models. The resulting artefacts are registered in the **Global Model Repository (GMR)**, making them directly reusable by other partners and system modules. This integration ensures that any IDS model—whether tree-based, linear, or deep learning—can benefit from reduced dimensionality and computational load without compromising detection performance.

For partners working with **large-scale or heterogeneous datasets**, SHAPRefine is particularly valuable: it lowers training cost, memory usage, and energy consumption, enabling deployment even in resource-constrained environments such as edge devices. At the same time, it boosts **per-class accuracy**, ensuring minority attack types are not overlooked. Thus, any component that requires efficient and robust classification can leverage SHAPRefine to achieve both **resource efficiency** and **high detection accuracy** in real-time 5G/6G scenarios.

6.3.2 SHAP-Based Incident Report & Analysis

The primary artefact of the Explainability Module is a detailed, SHAP-generated incident report in a structured JSON format. Following the model optimization, this service focuses on generating transparent, actionable insights. Using the refined model from the GMR, it produces the module's second key output: a detailed incident report that serves as an explainability metric. The goal of this report is to move beyond simple alerts and provide a comprehensive narrative that translates a model's complex decision-making process into transparent, human-readable, and actionable intelligence. It is designed to be consumed by both automated systems for orchestration and by human operators for in-depth analysis and final decision-making. This structured JSON report is the primary mechanism for communicating the model's reasoning and is crucial for **integration with other system components and partners**.

Instead of merely flagging a potential threat, the report provides a multi-layered explanation. It presents the model's final prediction alongside a full breakdown of its confidence across all potential outcomes, offering a nuanced view of the model's certainty. Critically, it quantifies the key pieces of evidence, the specific input features like traffic volume or connection states, that led to the conclusion. This evidence is then synthesized into a plain-language interpretation and a concrete recommendation, effectively telling the story behind the alert.

By providing human-understandable insights, this report serves as a vital bridge to other work packages. For the ZSM (WP4), it provides the necessary context for intelligent security orchestration. For the Trustworthiness Module (WP3), it offers the evidence required to validate the model's integrity. This integration is essential, as it ensures that explainability is not an isolated feature but a core enabler of system-wide trust.

The incident report metric is also model-agnostic and provides a multi-layered explanation. Most importantly, the key features highlighted in the report are drawn directly from the optimized subset identified during the refinement stage. This metric, registered in the **GMR** alongside the model it describes, closes the critical gap between raw detection and informed, effective response throughout the system.

6.3.2.1 Implementation details

The workflow begins when a single data instance, such as a network flow, is ingested. The pre-trained XGBoost model first generates a prediction and a full set of probability scores across all potential classes. Subsequently, the model's decision-making process for that specific instance is analyzed using the SHAP `TreeExplainer`, which quantifies the influence of each input feature. These quantitative outputs are then synthesized by a narration function, which generates a human-readable interpretation and a structured, two-part recommendation. Finally, an assembly function gathers all these components to produce the final incident report.

The primary consumption model for this module is a **file-based API**, which is well-suited for robust, asynchronous integration between different system components. For each analyzed incident, the module outputs a unique, self-contained `.json` file that can be ingested by a consuming service, such as the ZSM in WP4, on its own schedule. This design ensures a reliable hand-off of intelligence between work packages. For more tightly coupled integrations, the pipeline can be readily exposed as a REST API endpoint, allowing consumers to receive the JSON report directly in an HTTP response. The module's output has been validated using a hold-out test set containing various labeled attack types, confirming that the generated explanations and top features align with the expected characteristics of each threat.

As a concrete example, the report below was generated for a detected Denial of Service activity. It correctly identifies the event as a **"dos"** with a high confidence of **97%**. The explanation is supported by strong evidence from the top_features, which includes characteristics like is_GET_mthd and high src_ip_bytes, indicating a potential HTTP-based flood attack. The plain-language interpretation synthesizes this evidence, explaining that these features suggest a high-volume flood from a single source. Finally, the recommendation provides a clear, two-stage action plan for a security analyst, outlining both an **immediate** containment step (blocking the

source IP) and a **follow_up** investigation (analyzing the target service). This multi-layered report thus closes the critical gap between raw detection and an informed, effective response.


```
{
  "alert_id": "0a3e8652-d4b1-46a4-bd23-ad1ccfe1769a",
  "timestamp": "2025-09-18T11:04:47.678808+00:00",
  "prediction": "dos",
  "prediction_score": [
    {
      "key": "benign",
      "value": 1.3162882623873884e-06
    },
    {
      "key": "bruteforce",
      "value": 1.0718818884924985e-05
    },
    {
      "key": "ddos",
      "value": 0.029855886474251747
    },
    {
      "key": "dos",
      "value": 0.9700974225997925
    },
    {
      "key": "probe",
      "value": 2.2393542167264968e-05
    },
    {
      "key": "web",
      "value": 1.2278479516680818e-05
    }
  ],
  "uncertainty_indicator": false,
  "top_features": [
    {
      "feature": "is_GET_mthd",
      "influence": 0.7513051629066467
    },
    {
      "feature": "src_ip_bytes",
      "influence": 0.5472347140312195
    },
    {
      "feature": "conn_state_REJ",
      "influence": 0.29368096590042114
    },
    {
      "feature": "dst_bytes",
      "influence": 0.2839878797531128
    },
    {
      "feature": "is_file_transfered",
      "influence": 0.21455959975719452
    }
  ],
  "interpretation": "The model predicts a Denial of Service (DoS) attack. This is primarily driven by anomalies in 'is_GET_mthd' and 'src_ip_bytes', suggesting a high-volume flood from a single source targeting a specific service.",
  "recommendation": {
```

```
    "immediate": "Apply a network ACL to block the source IP at the firewall
or network edge.",
    "follow_up": "Analyze the target service for impact, check for signs of
compromise, and consider longer-term rate-limiting rules."
  }
}
```

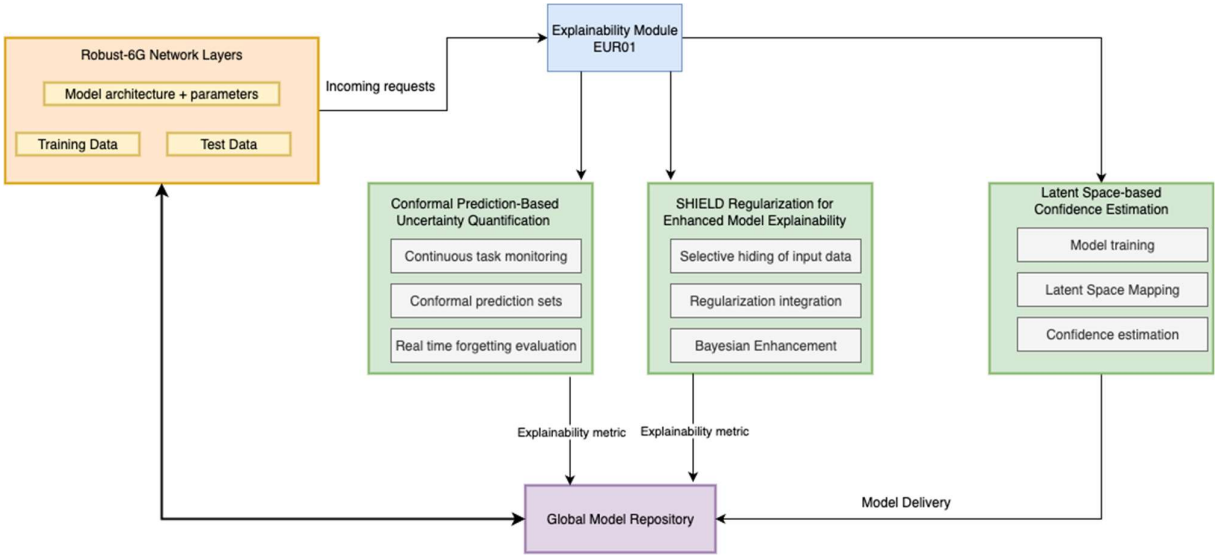

Figure 6-3 EUR explainability services

6.3.3 Conformal Prediction-Based Uncertainty Quantification

The Conformal Prediction Confidence Factor (CPCF) provides a model-agnostic approach capable of working with any baseline security model across diverse network data types. While theoretically applicable to any continual learning scenario, its initial implementation is specifically tailored for network security applications, where the input consists of evolving traffic patterns, threat signatures, and behavioural logs that require continuous adaptation without compromising historical knowledge retention.

The methodology leverages adaptive conformal prediction to systematically evaluate model confidence on previously learned tasks as new learning objectives are introduced. Through a dynamic monitoring strategy, tasks are continuously assessed for forgetting indicators, with confidence intervals providing early warning signals before significant performance degradation occurs. By focusing on uncertainty-aware metrics rather than simple accuracy measures, this approach enables proactive intervention in demanding network environments where maintaining consistent threat detection capabilities is critical. The primary output of this service is a continuously monitored model equipped with real-time forgetting assessment capabilities, providing network operators with transparent insights into model reliability across all learned tasks.

6.3.4 SHIELD Regularization for Enhanced Model Explainability

This module incorporates the SHIELD (Selective Hidden Input Evaluation for Learning Dynamics) regularization technique, augmented with Bayesian neural network capabilities through the IVON optimizer. This approach embeds explainability directly into the training process by selectively concealing portions of input data and measuring prediction discrepancies through Kullback-Leibler divergence. The methodology is fundamentally model-agnostic and applicable to various network security tasks, where understanding feature importance is critical for trust and reliability.

The SHIELD-IVON framework generates models that inherently provide uncertainty-aware explanations by maintaining both mean and variance estimates of network weights during training. This integration produces models with built-in explainability mechanisms that reveal which features are essential for accurate predictions while quantifying the confidence associated with these explanations.

6.3.5 Latent Space-based Confidence Estimation

This module incorporates a Variational Autoencoder architecture, which maps the input data into a structured latent space representation. Confidence can be assessed by leveraging the Mahalanobis distance between unknown samples and training data points within the latent spaced, providing a principled approach to uncertainty estimation. This framework, through this geometric approach, quantifies how well the new data patterns align with previously observed data. The primary output is a confidence estimation framework, which is bind to the model of a Variational Autoencoder, that delivers both classification results but also confidence scores based on the Latent Space proximity analysis.

6.4 Models and metrics stored in the Global Model Repository

Storing both deployable models and explanatory metrics within the GMR creates a transparent and auditable ecosystem where the performance of an AI model and the reasoning behind its decisions are formally linked and accessible. This approach is fundamental to building a trustworthy AI lifecycle, allowing partners and system components to not only consume high-performing models but also to understand and validate their operational behaviour. The following are key examples of such artefacts:

Models:

- The Optimized, Refined Model: This will be the output of the XAI-driven optimization service, which produces a deployable security model trained on a reduced, high-impact feature set. This model will be specific to the input of the service. Key characteristics of such an artefact include efficiency and robustness, which leads to less computational overhead and improved generalization.
- The Uncertainty-Quantified Model: This will be the output of the CPCF-based confidence estimation service, which produces a deployable model enhanced with conformal prediction capabilities for continual learning scenarios. This model will be specific to the operational context and learning trajectory of the input service. Key characteristics include adaptive confidence estimation and catastrophic forgetting resilience, which leads to improved reliability in dynamic environments and transparent performance monitoring across evolving tasks.
- The Regularization-Enhanced Model: This artefact is produced by the SHIELD-regularized training process, which generates a model optimized for explainability through selective hidden input evaluation. The model is trained to maintain consistent predictions even when portions of input features are concealed, resulting in improved generalization. Key characteristics include enhanced robustness to feature perturbations and built-in explainability mechanisms that reveal which features are truly essential for accurate predictions.
- The Latent Space-based Confidence Estimation Model: This is the output of a VAE model, a deployable model enhanced with confidence estimation for classification purposes. This is a model-specific framework, and its key characteristics include the proximity estimation of training and unknown data in the Latent Space. It provides a confidence metric correlated with the error of the model.

Metric:

- The Explainability Metric: Through a structured JSON incident report, essential context and transparency for any model stored in the GMR is provided using SHAP insights. Each report is a granular record of a model's reasoning for a specific prediction.

7 Concluding remarks

In this deliverable, the WP3 prototype has been presented. It is composed of four components: the **learning** module (presented in Section 3), the **trustworthiness** model (T, presented in Section 4), the **sustainability** module (S, presented in Section 5) and the **explainability** model (E, presented in Section 6), each responsible for a specific functionality. Model **learning** can be either centralized or distributed, and the final models are stored in a Global Model Repository (GMR), along with a list of *attributes* encompassing aspects T, S and E. If a model is not yet present in the GMR, it is trained on request. In Section 3.6, we presented usage examples for training and deploying a model with the proposed prototype. bubu

The following Table 7-1 and

Table 7-2 summarize the set of models and metrics that were produced within WP3 and that were registered in the GMR. These artifacts represent the tangible outputs of the Learning, Trustworthiness, Sustainability, and Explainability modules and serve as a foundation for enabling secure, efficient, and transparent AI services across the ROBUST-6G architecture.

The tables distinguish between models—deployable artifacts such as FL–based security models, energy-efficient architectures, and explainability-enhanced classifiers—and metrics, which capture essential evidence about model behaviour, robustness, energy efficiency, and interpretability. Presenting them side by side provides a clear overview of how the GMR not only stores model artefacts but also links them to verifiable metadata, ensuring consistency, accessibility, and trust for downstream consumers across the Robust-6G architecture.

It is important to note that the set of models and metrics presented here is not exhaustive. Rather, it reflects the current stage of the WP3 prototype as of this deliverable. The work has already undergone extensive research and integration, but we are continuing to evolve the prototype.

Table 7-1 Models registered in the Global Model Repository (GMR)

Artifact (GMR entry)	Module	Purpose / What it is	Key fields / KPIs captured
RF-fingerprinting FL model (baseline)	**Trustworthiness**	Vanilla FL model for RF-based identification/authentication/authorization.	Task, data refs, model arch, accuracy/F1, latency, size
Privacy-enhanced FL model	**Trustworthiness**	FL model trained with privacy mechanism (e.g., HE/DP) protecting client updates.	Accuracy/F1, privacy method+params, communication overhead, latency
Robust FL aggregated model	**Trustworthiness**	FL model aggregated with robust methods (e.g., Krum, Trimmed Mean, FoolsGold).	Accuracy/F1, robustness method, byzantine tolerance, training rounds
Adversarially-trained robust model	**Trustworthiness**	Model hardened via adversarial training to resist perturbations and backdoors.	Robust Accuracy RA(ε), clean accuracy, ε bounds, attack types used
Battery-aware model	**Sustainability**	Model produced under battery/renewable-aware scheduling.	Accuracy/F1, training energy, schedules used, participation stability
Decentralized ADMM model	**Sustainability**	Model trained with decentralized ADMM consensus for scalability.	Accuracy/F1, consensus iterations, communication rounds, convergence stats
SNN energy-efficient model	**Sustainability**	Spiking neural network for low-power, event-driven inference.	Accuracy/F1, inference energy, latency, SAR (spiking activity rate)
SHAPRefine optimized model	**Explainability**	Compact, high-accuracy model after XAI-driven feature selection.	Overall & minority-class Accuracy/F1, selected feature set, model size, latency
CPCF-enhanced model	**Explainability**	Model augmented with conformal prediction for uncertainty/forgetting monitoring.	CPCF scores, coverage/validity, drift/forgetting indicators
SHIELD-IVON regularized model	**Explainability**	Model trained with SHIELD regularization + IVON (Bayesian) for built-in explainability.	Mean/variance params, KL targets, robustness & explainability metrics
VAE latent-space confidence model	**Explainability**	VAE-based classifier or auxiliary estimator providing proximity-based confidence.	Accuracy, Mahalanobis thresholds, calibration metrics, latency

Table 7-2 Metrics registered in the Global Model Repository (GMR)

Artefact (GMR entry)	Module	What it measures / Why it matters	Key fields / KPIs captured
Poisoning detection outcomes	**Trustworthiness**	Flags/scores indicating suspected poisoned updates/models to protect aggregation.	Detector score, client IDs, round index, decision (clean/poisoned)
Robustness certificate pack	**Trustworthiness**	Formal robustness summary attached to a model.	RA(ε), OOD AUC, Backdoor ASR, eval dataset, date, tool version
Energy consumption (per-client/system)	**Sustainability**	Energy used during training/inference to guide sustainable decisions.	Joules per round/client, system totals, timestamps, HW profile
Computation vs communication energy ratio	**Sustainability**	Balance between compute and network energy to optimize scheduling/aggregation.	Ratio, absolute energy, communication bytes, model size
Learning performance (global)	**Sustainability**	Standard model performance over training runs.	Accuracy, F1, loss trajectory, convergence speed
Heterogeneity stability	**Sustainability**	Stability across heterogeneous clients to ensure fairness/robustness.	Std/var of client metrics, drop-out rate, fairness proxy
Scheduling stability & availability traces	**Sustainability**	Evidence of sustainable client selection and participation over time.	Selected clients per round, battery traces, link stats
Inference energy & complexity	**Sustainability**	Footprint of a deployed model at inference time.	Inference energy, latency, FLOAT/MAC counts, SAR (if SNN)
SHAP incident report	**Explainability**	Per-prediction transparency for human/machine consumption.	Prediction, scores, top SHAP features, narrative, recommendations
CPCF confidence/forgetting log	**Explainability**	Time-series of uncertainty & forgetting to monitor reliability.	Coverage, non-conformity scores, per-task confidence, alert flags
Latent-space confidence score	**Explainability**	Per-sample confidence based on latent proximity (VAE/Mahalanobis).	Distance, calibrated confidence, threshold, decision flag

In the next deliverable D.3.4, we plan to expand the GMR with more sophisticated models and richer metrics. The final prototype will thus provide an even more comprehensive and robust foundation for explainable, sustainable, and trustworthy AI in 6G systems.

References

[APM+25] Y. Abdennadher, G. Perin, R. Mazzieri, J. Pegoraro, and M. Rossi, "LightSNN: Lightweight Architecture Search for Sparse and Accurate Spiking Neural Networks," in Proc. of AMLSD 2025, 19-21 July 2025, Tokyo, Japan.

[ARC25] Y. Abdennadher, M. Rossi, and E. Cicciarella, "Convolutional Spiking-Based Gru Cell for Spatio-Temporal Data," in Proc. of IEEE MLSP 2025, 31 Aug.-3 Sep. 2025, Istanbul, Turkey.

[BDP+25] L. Ballotta, N. Dal Fabbro, G. Perin, L. Schenato, M. Rossi, and G. Piro, "VREM-FL: mobility-aware computation-scheduling co-design for vehicular federated learning," in *IEEE Transactions on Vehicular Technology*, vol. 74, no. 2, pp. 3311-3326, Feb. 2025, doi: 10.1109/TVT.2024.3479780.

[JP25] E. Jeong, and N. Pappas, "Battery-aware cyclic scheduling in energy-harvesting federated learning," in Proc. of IEEE SPAWC 2025, Surrey (UK), July 2025.

[KKF+25] L. Karaçay, F. Karakoç, R. Fuladi, "Privacy Protection Framework for Data Analytics in Management and Orchestration", Proceedings of 2025 International Symposium on Networks, Computers and Communications (ISNCC 2025).

[LC98] Y. LeCun and C. Cortes. *MNIST handwritten digit database*. 1998.

[MB+23] E. T. Martínez Beltrán, et al., "Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges," IEEE Communications Surveys and Tutorials, vol. 25, no. 4, pp. 2983–3013, Jan. 2023.

[Merkel14] D. Merkel. *Docker: lightweight Linux containers for consistent development and deployment*. *Linux Journal*, vol. 2014, no. 239, 2014.

[MSB+24] E. T. Martínez Beltrán, P. M. S. Sánchez, G. Bovet, B. Stiller, G. M. Pérez, and A. H. Celdrán, "Flighter: Decentralized Federated Learning and Situational Awareness for Secure Military Aerial Reconnaissance," IEEE Communications Magazine, 2024.

[MSL+24] E. T. Martínez Beltrán, P. M. Sánchez Sánchez, S. López Bernal, et al., "Mitigating communications threats in decentralized federated learning through moving target defense," Wireless Networks, vol. 30, pp. 7407–7421, 2024.

[Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[PBM+25] G. Perin, C. Bidini, R. Mazzieri, and M. Rossi, "ADMM-based Training for Spiking Neural Networks," *arXiv preprint arXiv:2505.05527,* 2025.

[RBG+22] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "NxTF: An API and compiler for deep spiking neural networks on Intel Loihi," in *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *18*(3), 1-22, 2022.

[ROB24-D32] ROBUST-6G "Initial Report on 6G Trustworthy and Sustainable AI Architecture and Requirements for Integrating Selected XAI Measures" https://robust-6g.eu/wp-content/uploads/2025/07/ROBUST-6G-D3_2_v1_0.pdf

[TON25] C. Lewis. *TON_IoT_network*. Hugging Face Datasets, 22 February 2025.

[TS25] GitHub for TorchServe, "A tool for serving and scaling PyTorch models in production", https://github.com/pytorch/serve

[WDS+20] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, et al. *Transformers: State-of-the-Art Natural Language Processing*. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.º

[Wolpe21] Z. C. Wolpe. “How to deploy Torch models with TorchServe”. Medium, 4 March 2021. Available: https://zachcolinwolpe.medium.com/how-to-deploy-torch-models-with-torchserve-d0bcbe12c9a1