ToRDIS

# D5.3: Final report on distributed AI and AI-based orchestration

Revision: v.1.1

| | |
|---|---|
| **Work package** | WP5 |
| **Task** | T5.1, T5.2, T5.3 |
| **Due date** | 31/10/2025 |
| **Submission date** | 12/11/2025 |
| **Deliverable lead** | Dušan Jakovetić (UNS), Lidija Fodor (UNS) |
| **Version** | 1.1 |
| **Authors** | Dušan Jakovetić (UNS), Lidija Fodor (UNS), Nemanja Petrovic (UNS), Miroslav Popović (UNS), Pavle Vasiljević (UNS), Claudia Soares (NOVA), Francisco Caldas (NOVA), Federico Metelo (NOVA), Sotirios Spantideas (NKUA), Anastasios Kaltakis (NKUA), David Vázquez Enríquez (GMV), Filippo Vannella (TID), David Solans Noguero (TID), Fionn Mc Inerney (TID), Miloš Simić (UNS) |
| **Reviewers** | Ping Hou (UOXF), Filippo Vannella (TID) |
| **Abstract** | This document represents the final report on the advances in the AI/ML primitives in T5.1, in the AI-driven orchestration in T5.2 and in the lightweight energy-efficient techniques in T5.3. The ML/AI tools are integral parts of the TaRDIS architecture. This document also provides descriptions of their connections to different aspects of TaRDIS, as well as the ML modelling of the TaRDIS use cases. Additionally, it contains a discussion on meeting milestones and objectives, while addressing corresponding KPIs in the context of AI/ML tools. |
| **Keywords** | decentralized machine learning and inference; AI/ML programming primitives; AI-driven planning, deployment and orchestration; lightweight and energy-efficient ML techniques |



Grant Agreement No.: 101093006
Call: HORIZON-CL4-2022-DATA-01

Topic: HORIZON-CL4-2022-DATA-01-03
Type of action: HORIZON- RIA

**Document Revision History**

| Version | Date | Description of change | List of contributor(s) |
|---|---|---|---|
| V0.1 | 16/06/2025 | Table of contents draft released. | UNS, NKUA, NOVA. |
| V1.0 | 31/10/2025 | Document ready for internal review | all authors |
| V1.1 | 6/11/2025 | Document reviewed internally | UOXF, TID |

## Disclaimer

Funded by the European Union

Funded by the European Union (TARDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

## Copyright notice



| Project funded by the European Commission in the Horizon Europe Programme | | |
|---|---|---|
| **Nature of the deliverable:** | **R** | |
| **Dissemination Level** | | |
| **PU** | *Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)* | ✔ |
| **SEN** | *Sensitive, limited under the conditions of the Grant Agreement* | |
| **Classified R-UE/ EU-R** | *EU RESTRICTED under the Commission Decision No2015/ 444* | |
| **Classified C-UE/ EU-C** | *EU CONFIDENTIAL under the Commission Decision No2015/ 444* | |
| **Classified S-UE/ EU-S** | *EU SECRET under the Commission Decision No2015/ 444* | |

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: datasets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

# EXECUTIVE SUMMARY

The TaRDIS project aims to develop a distributed programming toolbox that facilitates the development of decentralized, heterogeneous swarm applications across diverse operational settings. Work package 5 (WP5) is focused on enabling distributed artificial intelligence/machine learning (AI/ML) solutions and orchestration within TaRDIS, including: developing a framework that supports AI/ML programming primitives (Task 5.1), providing an artificial intelligence-driven (AI-driven) planning, deployment and orchestration framework (Task 5.2) and creating a library of lightweight and energy-efficient machine learning (ML) techniques (Task 5.3).

This deliverable presents the final results regarding distributed AI/ML and AI-based orchestration, through the tasks under WP5, built upon the previous contributions reported in Deliverable 5.1 (D5.1) [1] and Deliverable 5.2 (D5.2) [2]. Since the last deliverable, WP5 has achieved the planned progress across all tasks. This document reports on these advances, by describing the statuses of tools for each task. For Task 5.1 (T5.1), we describe the advances and final remarks regarding the following tools: Flower-based federated learning (FL), Python Testbed for Federated Learning Algorithms (PTB-FLA) and MicroPython implementation of PTB-FLA (MPT-FLA), and Fedra. For Task 5.2 (T5.2), we report on the PeersymGim environment and the Federated AI Network Orchestrator (FAuNO) statuses. In the context of Task 5.3 (T5.3), we discuss the achievements on advanced lightweight ML techniques, including pruning, early exit (EE) and knowledge distillation (KD), as well as the statuses of the Decentralized Early Exit Inference Tool (DEXIT). We also analyse the impact of WP5 tools from a broader perspective, through collaborations with other work packages, considering the whole TaRDIS architecture. Furthermore, we also discuss the final ML models for the TaRDIS use cases. Finally, this document provides an overview on the ways WP5 is addressing relevant milestones and objectives, as well as important Key Performance Indicators (KPIs).

Funded by the European Union

# Table of Contents

# LIST OF FIGURES



Funded by the European Union

# LIST OF TABLES

# ABBREVIATIONS

| | |
|---|---|
| **Adam** | Adaptive Moment Estimation (optimizer) |
| **AdamW** | Adam with Weight Decay (optimizer variant) |
| **AI** | Artificial Intelligence |
| **AI/ML** | Artificial Intelligence / Machine Learning |
| **API** | Application Programming Interface |
| **B** | Baseline (used in KPI identifiers) |
| **CIFAR-10** | Canadian Institute for Advanced Research 10-class image dataset |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **CSP** | Communicating Sequential Processes |
| **CSV** | Comma-Separated Values |
| **D** | Deliverable |
| **DDEKF** | Decentralized Deep Extended Kalman Filter |
| **DDPG** | Deep Deterministic Policy Gradient |
| **DEKF** | Deep Extended Kalman Filter |
| **DEXIT** | Decentralized Early-Exit Inference Tool |
| **DLR** | Deep Learning-based Reinforcement Learning |
| **DNN** | Deep Neural Network |
| **DP** | Differential Privacy |
| **DRL** | Deep Reinforcement Learning |
| **EE** | Early Exit |
| **EKF** | Extended Kalman Filter |
| **ESS** | Energy Storage System |
| **ETSI** | European Telecommunications Standards Institute |
| **FAuNO** | Federated AI Network Orchestrator |
| **FAuNO S** | FAuNO Standalone |
| **FedAvg** | Federated Averaging |

Funded by the European Union

| | |
|---|---|
| **FedBuff** | Federated Buffering |
| **FedProx** | Federated Proximal Optimization |
| **FedSGD** | Federated Stochastic Gradient Descent |
| **FL** | Federated Learning |
| **FLaaS** | Federated Learning as a Service |
| **FLOP** | Floating-Point Operation |
| **FRL** | Federated Reinforcement Learning |
| **FSn** | FAuNO Standalone Node |
| **Gelu** | Gaussian Error Linear Unit |
| **GPU** | Graphics Processing Unit |
| **GA** | General Assembly |
| **HUNOD** | Hybrid Unsupervised Outlier Detection |
| **HVAC** | Heating, Ventilation and Air Conditioning |
| **HTN** | Hierarchical Task Network |
| **HTTP** | Hypertext Transfer Protocol |
| **IDE** | Integrated Development Environment |
| **IJCAI** | International Joint Conference on Artificial Intelligence |
| **IoT** | Internet of Things |
| **JSON** | JavaScript Object Notation |
| **KD** | Knowledge Distillation (lightweight ML technique) |
| **KPI** | Key Performance Indicator |
| **LLM** | Large Language Model |
| **LQ** | Least Queues |
| **LSTM** | Long Short-Term Memory |
| **MAE** | Mean Absolute Error |
| **MARL** | Multi-Agent Reinforcement Learning |
| **MB** | Megabyte |
| **MLOps** | Machine Learning Operations |

| | |
|---|---|
| **ML** | Machine Learning |
| **MNIST** | Modified National Institute of Standards and Technology |
| **MPST** | Multiparty Asynchronous Session Types |
| **MS** | Milestone |
| **MSE** | Mean Squared Error |
| **MPT-FLA** | MicroPython-based Federated Learning API |
| **NeurIPS** | Neural Information Processing Systems Conference |
| **NUC** | Next Unit of Computing |
| **OM** | Orchestration Module |
| **P2P** | Peer-to-Peer |
| **PAT** | Process Analysis Toolkit |
| **pFedMe** | Personalized Federated Learning via Moreau Envelopes |
| **PINN** | Physics-Informed Neural Network |
| **PPO** | Proximal Policy Optimization |
| **PR** | Precision–Recall |
| **PTB-FLA** | Python-based TaRDIS Federated Learning API |
| **RAM** | Random Access Memory |
| **REST** | Representational State Transfer |
| **RGB** | Red–Green–Blue |
| **RL** | Reinforcement Learning |
| **RMSE** | Root Mean Squared Error |
| **RPI** | Raspberry Pi |
| **SCOF** | Synchronous Cooperative Offloading Framework |
| **SGD** | Stochastic Gradient Descent |
| **SL** | Split Learning |
| **SFL** | Split Federated Learning |
| **SoC** | State of Charge |
| **TCP** | Transmission Control Protocol |

Funded by the European Union

| | |
|---|---|
| **TO** | Task Offloading |
| **WP** | Work Package |
| **Wuw** | Wake-Up Word |

# 1 Introduction

## 1.1 Overview

Work package 5 is responsible for developing decentralized machine learning solutions, including AI/ML primitives (Task 5.1), AI-driven planning, deployment and orchestration (Task 5.2) and lightweight, energy-efficient ML techniques (Task 5.3). This document represents the final report on the results regarding Work Package 5 (WP5). It builds upon the previous WP5 deliverables. In D5.1 [1], we provided an initial overview and progress report for each task, the first ML modelling propositions for the TaRDIS use cases, as well as the initial relations with other work packages. D5.2 [2] contains the descriptions on the advances on each WP5 task, while recognizing the tools from these tasks as integral parts of the TaRDIS architecture. The document also provides more specific ML modelling approaches for the use cases, and deeper relations with other work packages. It also recognizes the relevant KPIs for the WP5 tools and discusses their contributions to WP5 and TaRDIS objectives.

In this report, we present the final outcomes and achievements within WP5. We demonstrate the capabilities of the tools from each task, while emphasizing the novelties since the last report (in D5.2 [2]). We highlight the placements of WP5 tools in the TaRDIS architecture and toolkit. This document also contains the final ML modeling approaches for each of the TaRDIS use cases. We depict the conformance of the WP5 results to milestones and objectives, and address the relevant KPIs for the tools. We also share some important aspects of collaboration and intersections with the rest of TaRDIS work packages. Finally, we summarize the main conclusions and outline future opportunities and visions.

## 1.2 Results summary

We first highlight the contributions in the context of Task 5.1, focusing on novelties since the previous report, D5.2 [2]. This task concerns the development of AI/ML primitives. We reported the development of 4 tools previously: The Flower-based FL tool (T-WP5-01/02/03), the Python Testbed for Federated Learning Algorithms (PTB-FLA) framework and the MicroPython implementation of PTB-FLA framework (MPT-FLA), i.e., T-WP5-04, and the Fedra framework (T-WP5-09). Now, we demonstrate the advances and the state of the art regarding these tools and approaches.

Second, we describe the contributions within Task 5.2, while emphasizing the advances during the last reporting period. This task is aimed at developing an AI-driven planning, deployment and orchestration framework. We introduced two important approaches earlier: the PeersimGym environment for addressing the task offloading problem, and the Federated AI Network Orchestrator (FAuNO) tool (T-WP5-05). We now present the novelties and the state of the art of these tools.

Finally, we focus on presenting the advances related to Task 5.3, underlining the novel contributions, since D5.2 [2]. This task is focused on lightweight and energy-efficient ML techniques. We introduced 3 such techniques in the previous deliverables: pruning (T-WP5-08), early exit (EE) of inference (T-WP5-06) and knowledge distillation (KD), i.e., T-WP5-07. Besides that, we also described the Decentralized Early Exit Inference Tool (DEXIT framework, that supports the deployment of the trained EE models in swarm nodes. In this report, we describe the novel properties and the state of the art of these approaches and tools.

Funded by the European Union

We also present the final ML modelling approaches for the TaRDIS use cases here, built upon the previously reported concepts, as well as contributions to different aspects of the projects, through collaborations with other work packages. We address the KPIs, corresponding to WP5 tools, and highlight contributions to project objectives and milestones.

## 1.3 Deliverable structure

The structure of this deliverable is as follows. In Section 1, we provide an introduction of the contributions and advances during the last reporting period. Section 2 is dedicated to present the advances regarding Task 5.1, specifically: the Flower-based FL tool (T-WP5-01/02/03), the PTB-FLA and MPT-FLA frameworks (T-WP5-04), and the Fedra framework (T-WP5-09). In Section 3, we describe the novelties in the context of Task 5.2, including the PeersymGym environment and the FAuNO orchestration tool (T-WP5-05). Section 4 contains the description of progress within Task 5.3, including novelties related to the developed lightweight techniques, i.e., pruning (T-WP5-08), early exit (T-WP5-06) and knowledge distillation (T-WP5-07), and the DEXIT framework. Section 5 is dedicated to highlighting the contributions of WP5 to different aspects of TaRDIS, by exploring the collaborations and connections with tasks from other work packages. In Section 6, we demonstrate the ML modeling of the TaRDIS use cases. Section 7 is meant to discuss the relevant KPIs and results of addressing them, as well as the contributions to relevant objectives and milestones. Finally, Section 8 provides conclusions about the work carried out within WP5. It summarizes the results and highlights the important perspectives.

Funded by the European Union

# 2 Advances on framework supporting AI/ML modelling primitives

This section is dedicated to contributions made regarding the development of AI/ML programming primitives. We discuss the advances related to the tools that were introduced in earlier deliverables (D5.1 [1] and D5.2 [2]), specifically, the Flower-based FL tool (T-WP5-01/02/03), PTB-FLA and MPT-FLA (T-WP5-04), and Fedra (T-WP5-09). The Flower-based FL tool (T-WP5-01/02/03) was already demonstrated in D5.2 [2], by explaining the user interface and novel approaches. Now, we describe the innovations made to the tool. Besides that, we discuss the preprocessing and inference/evaluation capabilities of the tool, as well as a new approach for anomaly detection, based on anomaly transformers. Regarding PTB-FLA and MPT-FLA (T-WP5-04), we present the newly developed PTB-FLA to Babel (T-WP6-04) adapter, as well as its adaptation for the GMV use case. We also demonstrate two additional novel approaches: the PTB-FLA based federated isolation forest algorithm and the translation of PTB-FLA algorithms into Communicating Sequential Processes (CSP) using ChatGPT. We also discuss the current state of the art and applications for the Fedra tool (T-WP5-09) in this section.

## 2.1 The Flower-based FL tool (T-WP5-01/02/03)

The Flower-based FL tool (T-WP5-01/02/03) represents a consolidation of three TaRDIS architecture tools (see D2.3 [3]): the Flower-based FL model training tool (T-WP5-01), the Data preparation for Flower-based FL model training tool (T-WP5-02) and the Flower-based FL model inference and evaluation tool (T-WP5-03). The Flower-based FL tool (T-WP5-01/02/03) offers a user-friendly approach for setting up the training process, by supporting the user choices by offering only applicable options. The tool has been recently adapted to become deployment-ready for real edge device clients. In this section, we discuss the novelties concerning the tool, including all three of its aspects (T-WP5-01, T-WP5-02 and T-WP5-03). In the context of FL model training (T-WP5-01), an important advancement is the development of a federated anomaly transformer-based anomaly detection algorithm, which we describe in detail here. We also provide a description of the preprocessing and inference/evaluation capabilities of the tool, that correspond to the data preparation (T-WP5-02) and FL model inference and evaluation (T-WP5-03) aspects. Additionally, we also report two novelties in methodological advances, that could be useful for directions that might emerge in the future, even beyond the boundaries of TaRDIS. One of them is considering high-probability convergence in online learning, in the presence of heavy-tailed noise [4], where a general framework of nonlinear Stochastic Gradient Descent (SGD) methods is considered. The other direction considers a family of distributed center-based clustering approaches [5], that work over connected networks of users.

The Flower-based FL tool (T-WP5-01/02/03) evolved during the past reporting period in several aspects. The first innovation is the transition from a simulation-based Flower federated training toward a clear separation between the client and server components of the tool. The tool is now deployment-ready and can easily be utilized in real settings. We run a large amount of tests on our cluster environment containing 16 nodes, 8 Intel i7 5820k 3.3GHz and 8 Intel i7 8700 3.2GHz Central Processing Unit (CPU) - 96 cores and 16GB DDR4 RAM/node, interconnected by a 10 Gbps network. We illustrate these tests in later sections, concerning the evaluations on ACT simulated data in Section 6.1, and the KPIs conformance in Section 7.2.1. We set different cluster nodes to be clients, while assigning one node to perform as an FL server. To support this setup, we created a script that starts the whole training on a cluster in one command, where the client and server approaches, datasets and number of clients are configurable. The script is available in union with the overall source code for the Flower-based FL tool (T-WP5-01/02/03) at the tool's GitHub

repository [6], under MIT licence. Although a cluster environment may not be the usual choice for FL, it serves as a good foundation for testing different aspects of the tools. However, as FL approaches often need to run on resource-constrained devices, we also provide some novelties supporting lighter execution. One aspect is introducing quantization to model parameters, in order to reduce transmission overhead, without losing accuracy (for more details, see Section 7.2.1). Besides that, as part of the collaboration within WP5, a KD-inspired algorithm has been developed and integrated into the tool. We describe it in more detail in Section 2.1.2.

The tool has also been expanded, by integrating the Anomaly Transformer-based anomaly detection approach [7] into our FL settings. This approach is described and evaluated in detail in Section 2.1.3 below. It proves the expandability of the tool, where even more complex models can be adapted to work in federated settings. Also, the autoencoder-based anomaly detection approach has been fully adapted to federated settings in the Flower framework as is now an integral part of it.

The Flower-based FL tool (T-WP5-01/02/03) has also been adjusted to support model retraining, where an existing, pre-trained model can be used to retrain in a new setup. Incremental training, or retraining, means continuing to improve a model by training it on new data, starting from a model that has already been pre-trained, rather than beginning from scratch. This allows the model to keep the already learnt knowledge and adapt it to new setups. In our tool, this process can work seamlessly, thanks to the underlying Flower framework [8]. The server begins by loading a previously trained global model from a file saved during an earlier training run. The pretrained model is sent to all clients, meaning that instead of initializing a new model, the clients train the received model on their local data. This process evolves the model, making it more robust in handling diverse data. This can be useful in federated settings, where new clients or data may appear over time. The approach may also save up computation and speed up convergence. Technically, we provide a flag that tells the server whether to start warm, from an existing model.

Regarding the preprocessing and inference/evaluation capabilities, the tool has also been widened in these aspects. We describe these properties in the following section (Section 2.1.1).

### 2.1.1 Preprocessing (T-WP5-02) and inference/evaluation (T-WP5-03)

As stated before, the Flower-based FL tool also contains a preprocessing (T-WP5-02) and an inference/evaluation module (T-WP5-03), beside the main FL training module (T-WP5-01). We now provide an overview of the preprocessing and inference/evaluation facilities.

The preprocessing module (T-WP5-02) provides a set of data preparation possibilities. It can handle input data in JavaScript Object Notation (JSON) format and parse and flatten it into a pandas DataFrame. Additionally, the tool can apply timestamp normalization and feature extraction as well. It also has an additional splitting feature that is useful for testing. Often, the test datasets come unified, as one file. The preprocessing tool enables splitting into an arbitrary number of clients. Also, it can split each client dataset into train, test and validation subsets, by specifying the desired percentages for each. The tool can be used in a centralized manner, for testing, when preparing the whole dataset and splitting it to clients. However, it can also be used in real settings, on clients, where each client can apply the needed preprocessing steps on its own data. We illustrate the usage of this module on the synthetic ACT dataset. This module is highly extensible, so that additional preprocessing and preparatory approaches can be easily integrated.

The inference/evaluation module (T-WP5-03) was already partially illustrated in D5.2 [2], where we showed that the tool produces outcomes that are appropriate for the particular setup. We showed the case where this means plotting the accuracies and/or losses over the training rounds. A trained model can be used for inference on arbitrary data and it can be evaluated according to the task being solved. We illustrate some of the aspects of this module through this document, while applying our tool to various datasets for solving different tasks. For instance, inference occurs whenever the trained models, such as autoencoder or transformer-based anomaly detection are applied to validation or test data to generate outputs like reconstruction errors, anomaly scores, or predicted labels, which is illustrated in Section 6.1. These outputs represent the responses of the model to unseen data and are the basis for further analysis. Evaluation builds upon these results, by measuring and interpreting the performance of the model, using quantitative metrics, such as accuracy, precision, recall, F1-score, losses, thresholds. Additionally, our tool includes evaluations of system-level indicators, such as training latency communication overhead and memory usage (see Section 7.2.1). These reflect the efficiency of the FL process. These aspects together demonstrate the predictive effectiveness and the operational performance of the Flower-based FL tool (T-WP5-01/02/03) across different model architectures and test scenarios. Additionally, the preprocessing module has the ability to introduce anomalies in a non-anomalous dataset, for testing purposes. We illustrate this approach in Section 6.1, in the context of the ACT use case.

The three modules building the Flower-based FL tool (T-WP5-01/02/03) are tightly connected and meant to be used together. The data preparation module (T-WP5-02) can help to transform the input data to a form that is suitable for the training process, while the inference/evaluation module (T-WP5-03) enables applying the trained model and gathering meaningful insights.

### 2.1.2 Knowledge distillation inspired algorithm

The algorithm inspired by knowledge distillation was developed by combining the logic of Federated Averaging (FedAvg) [9] and knowledge distillation [10]. The idea emerged through collaboration among the different working groups within WP5, particularly the team developing the Flower-based FL tool and the team working on lightweight ML approaches.

On the server side, the process remains unchanged. However, on the client side, during the local training phase, the main idea is to leverage a pre-trained, more complex teacher model to assist in training a simpler student model. This results in a lightweight model that requires less computational effort during inference, yet benefits from the guidance of a larger and more reliable model.

The approach was tested on the Canadian Institute for Advanced Research 10-class image dataset (CIFAR-10 dataset), using ResNet-18 as the global teacher model, while a simple convolutional neural network (CNN) served as the student model.

Table 1 below summarizes the results across ten communication rounds with 5 clients, comparing performance of CNN (FedAvg), the Resnet-18 (FedAvg), and the CNN with the help of ResNet18 (FedAvg+KD). The results show a consistent 2-3% improvement in accuracy each round, with the FedAvg+KD outperforming the baseline FedAvg in every round. After 10 rounds, the FedAvg + KD CNN reached approximately 73.2% accuracy, compared to 71.2% for the baseline FedAvg CNN. The FedAvg ResNet-18 reached an accuracy of around 84.5%.

To clarify the computational difference between the teacher and student models, we provide model size details. The student CNN model contains approximately 4.2 million trainable

parameters, while the teacher ResNet-18 model has around 11.2 million. This makes the student model roughly 2.5 times smaller.

TABLE 1: MODEL COMPARISON REGARDING THE USAGE OF KD-INSPIRED APPROACH

| Round | CNN_Acc | CNN+KD_Acc | ResNet_Acc |
|---|---|---|---|
| 1 | 0.4682 | 0.4607 | 0.55 |
| 2 | 0.5774 | 0.5967 | 0.63 |
| 3 | 0.6355 | 0.6448 | 0.7 |
| 4 | 0.6613 | 0.674 | 0.75 |
| 5 | 0.6808 | 0.6995 | 0.79 |
| 6 | 0.6972 | 0.709 | 0.81 |
| 7 | 0.7039 | 0.7186 | 0.825 |
| 8 | 0.7121 | 0.7276 | 0.835 |
| 9 | 0.7085 | 0.7341 | 0.842 |
| 10 | 0.7125 | 0.7325 | 0.845 |

Figure 1 below presents the corresponding learning curves, showing the accelerated convergence of the FedAvg + KD CNN approach compared to the baseline FedAvg CNN. The inclusion of KD thus enhances the learning process and final accuracy, confirming the suitability of the method for resource-constrained FL scenarios.

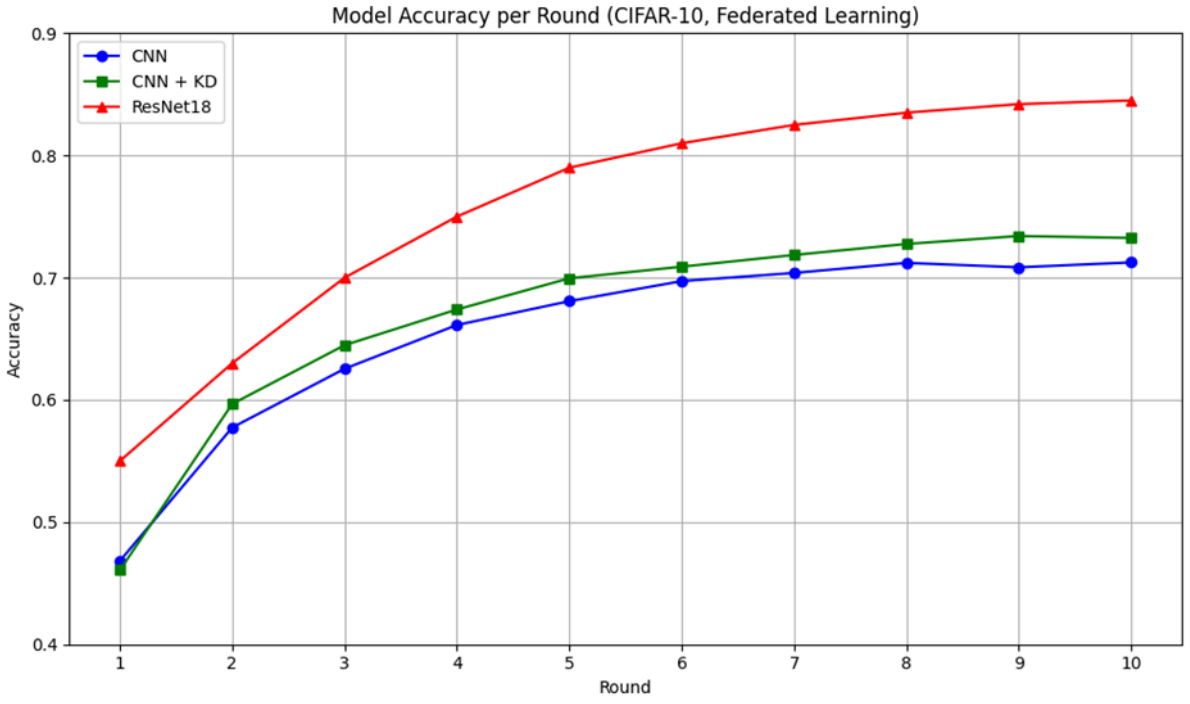


FIGURE 1: MODEL ACCURACY FOR THE KD-INSPIRED APPROACH

### 2.1.3 Transformer-based anomaly detection

The Anomaly Transformer [7] is a model designed for unsupervised anomaly detection in time series data. It leverages the Transformer architecture to model both pointwise and pairwise relationships, with the key innovation being the concept of Association Discrepancy.

Prior-association (P) is a learnable Gaussian kernel that computes prior attention scores based on temporal distance. Series-association (S) is standard scaled dot product attention that captures associations with the entire sequence. The core idea of anomaly detection in this model is to measure the discrepancy between prior and series associations.

$$\mathrm{AssDis}(P,S,X) = \left[\frac{1}{L}\sum_{m=1}^{L}\left(\mathrm{KL}(P_{i,:}^{m}\|S_{i,:}^{m}) + \mathrm{KL}(S_{i,:}^{m}\|P_{i,:}^{m})\right)\right]_{i=1,\ldots,N}$$

where KL is Kullback-Leibler divergence, and L is the number of layers in the model. Due to the rarity of anomalies, nontrivial associations from abnormal points to the whole series are difficult. Thereby, the anomalies' associations will be stronger on their adjacent time points and make them more identifiable.

#### 2.1.3.1 Centralized approach

In Deliverable D5.2 [2], we presented the centralized anomaly detection framework based on the Hybrid Unsupervised Outlier Detection (HUNOD) method [11] and reported its achieved performance. Now, this approach has been adapted to federated settings, in the context of the Flower framework. In this section, we extend that study by introducing and evaluating the Anomaly Transformer model [7], which leverages self-attention mechanisms to capture temporal dependencies and contextual deviations within the time series.

The Anomaly Transformer was included to enable a comparative analysis between classical reconstruction-based approaches (as in HUNOD) and attention-based anomaly detection methods, providing a systematic evaluation of model robustness and generalization under both clean and noisy label conditions.

##### 2.1.3.1.1 Experimental setup

All experiments were conducted on the MetroPT-3 dataset [12], a multivariate time series comprising 14 features per time step, collected from analogue and digital sensors monitoring a train compressor over seven months. The dataset includes both normal and faulty operating conditions. It was divided into 4800 training and 1200 testing samples, with 50 % of the test set labeled as anomalous (600 anomalies and 600 normal samples).

Besides the previously assessed HUNOD model, which was trained to reconstruct normal samples and detect anomalies via reconstruction error, this section also evaluates the Anomaly Transformer, a transformer-based model that captures both pointwise and pair-wise temporal relations through the concept of association discrepancy. Model performance was evaluated using accuracy, precision, recall, and f1-score metrics.

##### 2.1.3.1.2 Centralized anomaly detection with exact labels

The results of the first experiment are summarized in Table 2, where the training data consisted exclusively of normal samples, and the testing set contained an equal proportion of normal and anomalous samples.

Table 2: Performance of Autoencoder and Anomaly Transformer under exact labels

| Model | Precision | Recall | Accuracy | F1-score |
|---|---|---|---|---|
| Autoencoder | 0.9981 | 0.8833 | 0.9408 | 0.9372 |
| Anomaly transformer | 0.9901 | 0.9983 | 0.9942 | 0.9942 |

The Anomaly Transformer outperformed the Autoencoder across nearly all metrics, particularly in recall, accurately detecting the majority of true anomalies while maintaining high precision. In contrast, the Autoencoder, although highly precise, failed to detect certain anomalies, leading to lower recall and overall accuracy.

#### 2.1.3.1.3 Centralized anomaly detection with inexact (flipped) labels

In the second experiment, the model robustness was evaluated by introducing label noise into the training data. Specifically, 10% of normal samples were mislabeled as anomalous (inexact anomalies), and 10% of true anomalies were mislabeled as normal (inexact non-anomalies). Consequently, the training set contained 90% normal and 10% mislabeled samples, while the test set included 50% true anomalies, 40% normal samples, and 10% inexact anomalies.

The results presented in Table 3 show that the Autoencoder's performance decreased significantly in the presence of label noise, indicating a high sensitivity to imperfect data. Conversely, the Anomaly Transformer demonstrated strong robustness, maintaining an F1-score above 0.98 even with 50% of the labels flipped. These findings confirm its superior generalization capability and resilience to uncertain or noisy data.

Table 3: Performance of Autoencoder and Anomaly Transformer under inexact labels

| Model | Precision | Recall | Accuracy | F1-score |
|---|---|---|---|---|
| Autoencoder (10% flipped) | 0.9619 | 0.7361 | 0.8241 | 0.8340 |
| Anomaly transformer (10% flipped) | 0.9901 | 1.0000 | 0.9950 | 0.9950 |
| Anomaly transformer (50% flipped) | 0.9740 | 1.0000 | 0.9867 | 0.9868 |

Our experimental analysis demonstrates that the Anomaly Transformer consistently outperforms the Autoencoder across all evaluation metrics and experimental conditions. Its attention-based temporal modeling enables more accurate anomaly localization and deeper contextual understanding. Furthermore, the model exhibits strong robustness to label noise, maintaining near-perfect recall and F1-score even when 50% of the labels are flipped.

Compared to the Autoencoder-based approach previously reported in Deliverable D5.2 [2], the Anomaly Transformer provides clear advantages in anomaly detection accuracy and robustness to data uncertainty. Overall, this analysis confirms that it is a promising candidate for further application in distributed anomaly detection frameworks.

Funded by the European Union

#### 2.1.3.2 Distributed approach

The described Anomaly Transformer approach was integrated into the Flower-based FL tool (T-WP5-01/02/03), enabling decentralized, federated execution. The implementation of the Anomaly Transformer was tested within our Flower FL tool using the same dataset employed by the original authors for evaluation. The dataset was partitioned into five distinct local subsets, and the FedAvg algorithm was executed. The results presented in Table 4 correspond to 10 communication rounds with one local training epoch per client. The comparison presents the final rounds for both centralized and decentralized approaches.

TABLE 4: PERFORMANCE COMPARISON OF THE CENTRALIZED AND DECENTRALIZED ANOMALY TRANSFORMERS

| Metric | Centralized | Federated | Delta |
|---|---|---|---|
| F1-score | 97.89% | 96.08% | -1.81% |
| Precision | 96.91% | 94.29% | -2.62% |
| Recall | 98.90% | 97.93% | -0.97% |

It is evident from Table 4 that the differences between the performance of the centralized and federated implementations are minor. These findings demonstrate that even complex models, such as the Anomaly Transformer, can be successfully integrated into our framework. Consequently, the process of training and evaluating such advanced architectures becomes a streamlined task, reduced essentially to selecting configuration options and providing and preparing the appropriate data.

## 2.2 PTB-FLA AND MPT-FLA (T-WP5-04)

PTB-FLA (T-WP5-04) is already integrated into the TaRDIS toolbox (see D3.4 [13]) and MPT-FLA is planned for integration by D3.6. Generic FL algorithms of PTB-FLA and MPT-FLA were formally verified, see Section 5.3.

PTB-FLA (T-WP5-04) is used in the GMV use case and will be evaluated as an integral part of the GMV use case in D7.3, see Section 6.2. Additionally, PTB-FLA and MPT-FLA (T-WP5-04) were already evaluated as stand-alone tools, see Section 7.2.

In the rest of the section, we present the four main PTB-FLA and MPT-FLA (T-WP5-04) advances since D5.2 [2]: (1) PTB-FLA to Babel (T-WP6-04) adapter based on data dissemination service, (2) PTB-FLA to Babel (T-WP6-04) adapter for GMV use case, (3) PTB-FLA based Federated Isolation Forest algorithm, and (4) a solution for translating PTB-FLA algorithms into CSP processes using ChatGPT. The next subsections describe each of these four advances.

### 2.2.1 PTB-FLA (T-WP5-04) to Babel adapter based on data dissemination service

In this section we provide an overview of PTB-FLA (T-WP5-04) to Babel (T-WP6-04) adapter, which aims to allow PTB-FLA based FL applications to run on multiple hosts without any changes to already tested and evaluated code.

To facilitate communication between FL nodes, PTB-FLA (T-WP5-04) leverages abstractions from Python's multiprocessing module, enabling efficient inter-process message passing on a single host. This way of communication is however not well suited for communication over the network. In order to facilitate network communication with minimal setup, Babel with its membership discovery and data dissemination based on gossiping protocols was chosen.

The PTB-FLA to Babel adapter (Figure 2) is composed of two key components: Doppelganger instances, which emulate remote PTB-FLA applications, and the BabelAdapterApp, which manages membership and message dissemination using network protocols provided by the Babel framework. Communication between these components is conducted over Transmission Control Protocol (TCP), with messages serialized in JSON format to ensure interoperability between Python and Java. Each Doppelganger interacts with its corresponding local PTB-FLA instance via inter-process communication (IPC) primitives provided by Python's multiprocessing module.
A Doppelganger instance represents a proxy of a remote PTB-FLA node. It bridges Python's multiprocessing communication mechanisms with the BabelAdapterApp, facilitating message delivery to the corresponding PTB-FLA instance on the network.

The BabelAdapterApp utilizes communication primitives such as an eager-push gossip broadcast mechanism to facilitate interaction between devices within a local network. It handles both device discovery and end-to-end message dissemination, ensuring reliable and seamless communication among all participants. This communication is based on Babel's Data Dissemination service.

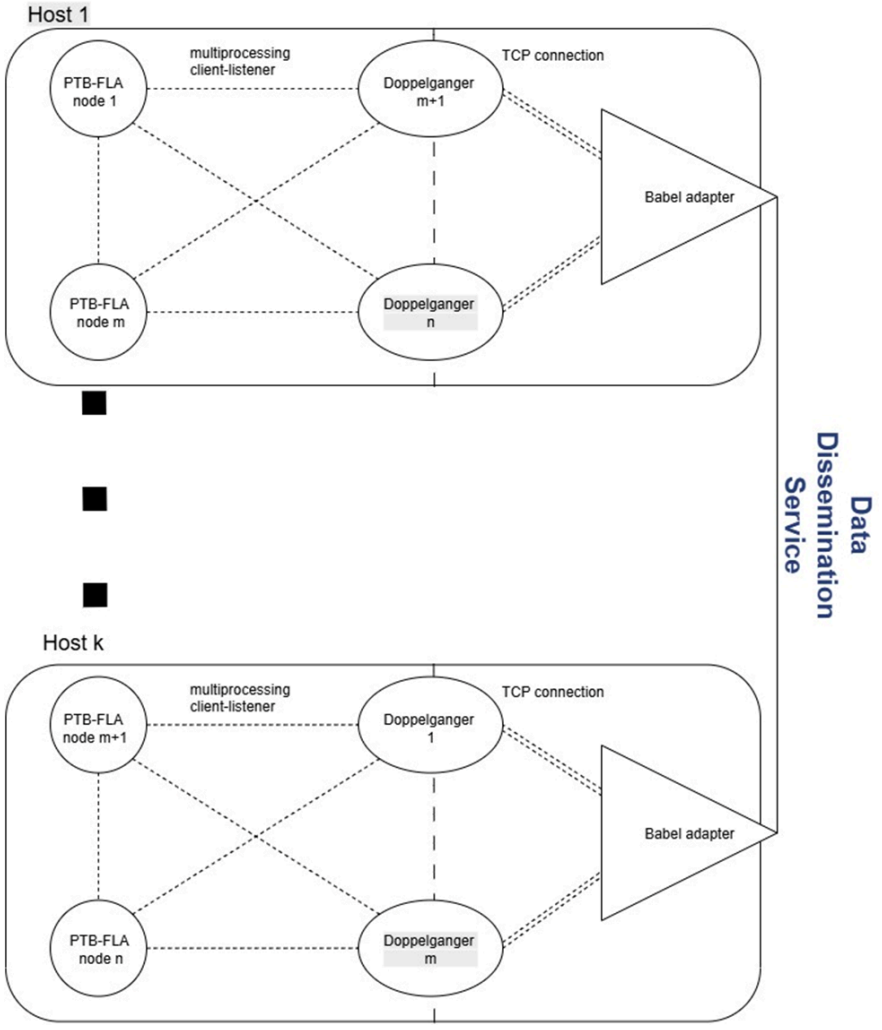


FIGURE 2: PTB-FLA TO BABEL ADAPTER ARCHITECTURE

The development of the PTB-FLA (T-WP5-04) to Babel adapter enables seamless integration of PTB-FLA applications with networked environments managed by Babel (T-WP6-04). As a result, PTB-FLA instances running on separate hosts can participate in network communication, without requiring modifications to either the Application developed in PTB-FLA (T-WP5-04) or the PTB-FLA itself. The adapter ensures reliable message delivery, bridging Python-based multiprocessing communication with Java-based networking implemented using Babel. This abstraction simplifies deployment, supports cross-host federated learning experiments, and lays the foundation for simple scaling PTB-FLA (T-WP5-04) across the network.

### 2.2.2 PTB-FLA (T-WP5-04) to Babel adapter for GMV use case

In this section, we present the PTB-FLA (T-WP5-04) to Babel (T-WP6-04) Adapter as part of the GMV use case (Figure 3). Adapter developed for the GMV use case is based on the core principles of the Adapter outlined in Section 2.2.1.

As in the previous version of the Adapter, it is composed of two key components: **Doppelganger** instances and **BabelAdapterApp**. Doppelgangers communicate with PTB-FLA instances via multiprocessing, while communication between Doppelgangers and the adapter app has been upgraded from TCP to Hypertext Transfer Protocol (HTTP)

protocol. The interface through which the doppelgangers interact with the Babel portion of the adapter is the “/messages” endpoint.

In addition to this, PTB-FLA (T-WP5-04) has been modified to support interaction with HTTP based services. In the case of the GMV use case adapter, these are the membership service and the visibility model, which are accessible through the “/membership” and “/visibilityModel” endpoints respectively. This allows for increased flexibility and leveraging services required in order to cater the use case specific needs.

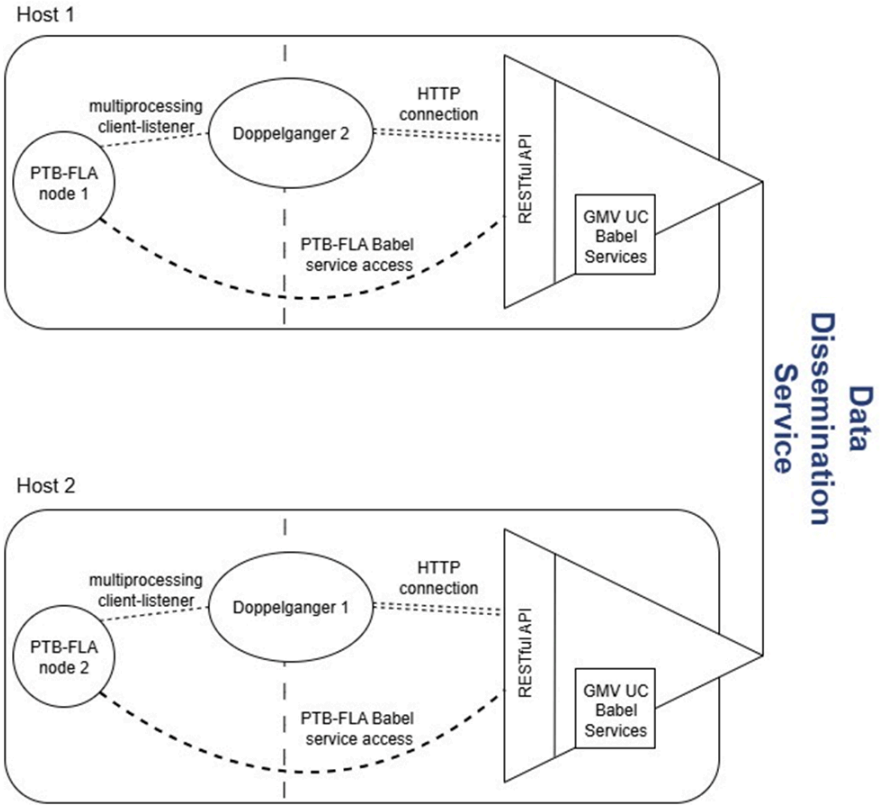


FIGURE 3: GMV USE CASE ADAPTER ARCHITECTURE

From the FL application developers’ side, code they implemented remains unchanged while requiring minimal setup (i.e., network interface information and which instances are remote). This aids the correct by construction (i.e., composition) approach followed in the GMV use case ensuring correctness of the whole system by composition of well-tested components.

The PTB-FLA (T-WP5-04) to Babel Adapter for GMV Use Case strengthens system modularity, preserves correctness by maintaining clear separation between components, while providing an interface between use case components.

### 2.2.3 PTB-FLA (T-WP5-04) based federated isolation forest algorithm

In this section, we provide a short digest of research that is to be published by IEEE Xplore in the ZINC 2025 proceedings (publicly available as a preprint [14]), which comprises the problem formulation and its solution, as well as this research achievements.

In the following, we present the evaluation of the Federated Isolation Forest (PFLiForest) algorithm developed using PTB-FLA (T-WP5-04), and discuss the evaluation results as they pertain to relevant TaRDIS Key Performance Indicators (KPIs).

**Problem formulation and its solution.** The training of federated anomaly detection models in resource-constrained environments introduces challenges related to training time, development overhead, and algorithm synchronization. While traditional isolation forests offer computational efficiency and unsupervised learning, their direct application in federated settings is non-trivial due to the complexity of model aggregation. These factors impact both system responsiveness and development effort.

To address these issues, this research introduced a new algorithm, PFLiForest, a federated implementation of the Isolation Forest algorithm trained in a layer-by-layer manner. The training process was restructured into an iterative, queue-based mechanism, and enhanced with custom phase synchronization to ensure deterministic training flow across all nodes. This layer-by-layer approach ensured correct termination of the model training process, while conforming to the federated learning ideals of knowledge sharing across nodes while maintaining training data privacy.

Use of PTB-FLA (T-WP5-04) framework in the algorithm development provided the following benefits: (1) having a clear development roadmap reducing time needed to verify algorithmic correctness in comparison to the baseline sequential implementation of Isolation Forest, and (2) abstracting away node to node interactions between the FL participants bringing the algorithm logic into focus and circumventing the usual technical issues that may arise when developing a distributed application (i.e., communication coordination, startup of FL participants, etc.). These benefits lead to less complex development and reduced implementation time which although it has not been explicitly measured aligns with the motivations behind K-O-1.3.

**Research achievements.** The primary research contributions can be summarized as follows: (1) Introduction of PFLiForest, a practical implementation of FLiForest adapted for PTB-FLA (T-WP5-04). (2) Experimental Validation: The algorithm was deployed on real temperature sensor data in a simulated federated setup involving edge devices. It demonstrated high anomaly detection performance with Area Under the Receiver Operating Characteristic curve (AUC-ROC) above 0.999 and area under the precision recall curve (AUC-PR) above 0.97 under memory constraints (<160 KB), confirming its suitability for constrained environments. (3) Efficiency Analysis: Comprehensive evaluation of trade-offs between model complexity (tree depth, forest size), resource usage (memory footprint, execution time), and detection performance. The results confirm that execution time grows almost linearly with memory usage, which in turn is influenced by the number of trees, tree depth, and training data volume. The results of the FL training time are shown below in Figure 4. The key takeaway is that the optimal configuration identified earlier (depth=6, trees=25, data=200) not only fits within the memory constraints of the target device (Raspberry Pi Pico) but also meets acceptable training time requirements, as shown in Figure 4, which aligns with K-O-2.5. This reinforces the suitability of the selected configuration for efficient on-device training in real-time, resource-constrained edge environments. (4) Comparative performance evaluation: When compared with the standalone iForest algorithm, PFLiForest achieved near-equivalent detection accuracy while preserving data privacy. Despite minor degradation in AUC-PR due to data locality, the performance gap narrowed as model complexity increased—highlighting the feasibility of federated approaches for real-world deployments.

Funded by the European Union

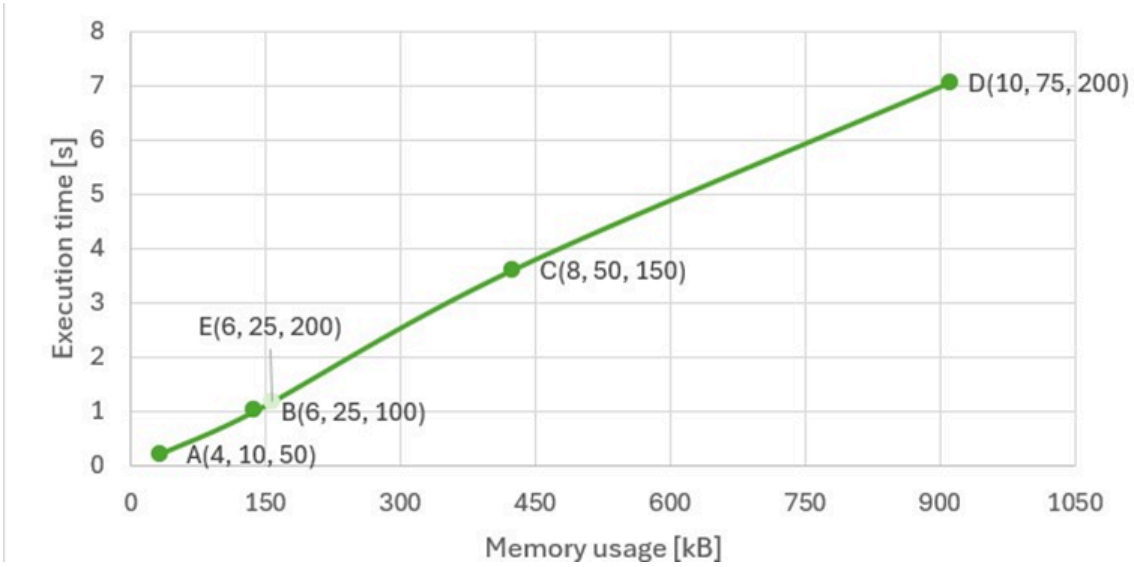


FIGURE 4: PFLIFOREST TRAINING TIME AGAINST MEMORY USAGE

The main advantage of this research over traditional isolation forest implementations is that it enables federated training across distributed edge devices, without requiring access to raw sensor data or centralized storage. The PFLiForrest also represents a deployment-ready implementation of Federated Isolation forests aimed at edge devices running MicroPython.

The main limitation of this research is the high communication overhead inherent in the iterative training process, which may lead to bursty traffic and increased latency during tree construction. The main direction for future work is to optimize communication, for example through batching or local caching strategies, while preserving synchronization semantics and maintaining the accuracy of global model aggregation.

### 2.2.4 Translating PTB-FLA (T-WP5-04) algorithms into CSP processes using ChatGPT

In this section, we provide a short digest of the research that is to be published by IEEE Xplore in the MIPRO 2025 proceedings (publicly available as a preprint [15]), which comprises the problem formulation and its solution, as well as key research achievements.

**Problem formulation and its solution**. The PTB-FLA FL orchestration protocols were formally verified in [16] by using the process algebra Communicating Sequential Processes (CSP) and the model checker Process Analysis Toolkit (PAT). The main limitation of [16] is that Python implementations of PTB-FLA FL orchestration protocols were manually translated into the corresponding CSP processes using an ad-hoc approach. Motivated by the goal to overcome this limitation, the authors of [17] devised a process in which they: (1) first rewrite the original Python code using the restricted actor-based programming model, and then (2) systematically construct CSP code from the rewritten Python code. This research was motivated by the idea to see whether it would be possible to skip rewriting the Python code, and to directly translate the original Python code into the corresponding CSP processes using contemporary Large Language Models (LLMs) like ChatGPT.

As a solution, a translation process is introduced wherein ChatGPT is used to automate the translation of the federated learning algorithms in Python into the corresponding CSP processes. The translation process is a simple iterative process, which comprises the

following steps: (1) specify the context (i.e., the prompt), (2) ask ChatGPT to complete the assignment, (3) test the output CSP# code by the model checker PAT, (4) if the CSP# code successfully passes testing, stop, else if the CSP# code is close to what is expected and contains a small number of simple errors, manually modify the code and return to step 3, else return to step 1. To successfully pass testing by PAT, the output CSP# code must pass two steps: (3.1) it must be syntactically correct and (3.2) all the specified (desired) system properties must be successfully verified (PAT uses the term validated).

The authors tried to create minimal contexts to minimize the overall translation effort/cost (because the ChatGPT service is charged by the context size). For the sake of this research, to get an estimation of the context quality (minimality), after the translation process was successfully completed, they asked ChatGPT for feedback by asking the following three questions: (i) On a scale from 1 to 10, where 10 is the hardest, how would you rate this assignment? (ii) What was the most important part of the context that you used to complete the assignment? and (iii) What was the redundant part of the context?

The proposed translation process was experimentally validated by successful translation (verified by PAT) of both PTB-FLA (T-WP5-04) generic centralized and decentralized federated learning algorithms (for more details see Section III in [15]).

**Research Achievements**. The main research achievements are the following: (1) the automated translation process for translating PTB-FLA generic federated learning algorithms in Python into CSP processes, (2) the input contexts (prompts) given to ChatGPT, (3) the CSP models of both PTB-FLA generic centralized and decentralized FL algorithms (FLAs) produced by this translation process, and (4) the feedback questionnaires completed by ChatGPT that are used to estimate the minimality of the used contexts.

The main advantage of this research over the previous papers [16], [17] is that part of the work is done by ChatGPT. Another advantage over [17] is that this approach accepts the original Python code as input, whereas [17] requires the code to be rewritten in a restricted actor-based programming model. The main direction of the future work is to overcome the need for manual correction of the errors made by ChatGPT.

## 2.3 FEDRA

Fedra (T-WP5-09) is a fully decentralized FL framework to enable the intelligence sharing between swarm nodes in a Peer-to-Peer (P2P) manner. Its architecture and preliminary results have already been described in the previous deliverable D5.2 [2], along with development and implementation considerations. The final version of the Fedra tool (T-WP5-09) can be found in its GitHub repository [18], where no major updates were applied during the last year of the project. In this deliverable, we report the measured KPIs related to the operation of the Fedra tool (T-WP5-09) in Section 7.2.4.

Regarding the configuration, the following parameters for the dataset, the model and the training process were utilized for measuring the KPIs:

***Dataset specifications***

Dataset: Household Energy Consumption Data (60-minute intervals)

- File: household_data_60min_singleindex.csv
- Households: 2 residential units (DE_KN_residential3 & DE_KN_residential4)
- Duration: 487 days (~16 months)
- Resolution: Hourly measurements

- Total Samples: ~11,688 per household
- Feature: Grid export measurements (energy exported to grid)

Task: Time series regression - Predict next hour's grid export based on previous 336 hours (14 days)

***Model Architecture***

Type: Stacked Long Short-Term Memory (LSTM) Regressor

- 4 LSTM layers (50 hidden units each)
- Dropout rate: 0.2 after each layer
- Input: 336 timesteps × 1 feature
- Output: 1 value (next hour prediction)
- Total parameters: ~51,000

***Training Setup:***

- Batch size: 64
- Local epochs: 2 per round
- Training rounds: 2
- Optimizer: Adaptive Moment Estimation, i.e. Adam (learning_rate=0.001)
- Loss: Mean Squared Error (MSE)

# 3 Advances on AI-driven planning, deployment and orchestration framework

In this section, we describe the state of the art regarding the AI-driven planning, deployment and orchestration aspects of the AI/ML component of TaRDIS. First, we outline the advances on the PeersimGim environment. Next, we describe the advances on the FAuNO (T-WP5-05) orchestrator, which can now operate outside PeersimGym and represents a pluggable orchestrator to Babel (T-WP6-04).

## 3.1 PeersymGim: An environment for solving the task offloading problem with reinforcement learning

Since D5.2 [2], development of PeersimGym has continued with utilities added to adjust task dataset sizes for different device classes and additional topologies designed to evaluate various components. We have also begun developing a wrapper for the PeersimGym environment that introduces an event-based reward function, with optional integration of the original reward as a shaping mechanism.

The new reward at time step $t$ is:

$$R_t = U_t - D_t + F_t$$

where

- $U_t$: utility from completed tasks
- $D_t$: penalty from dropped tasks
- $F_t$: adjustment based on the reward shaping term

Utility and Penalty are defined as: $U_t = N_{\text{fin}} \cdot \alpha$, and

$D_t = N_{\text{fail}} \cdot \alpha$, where

- $N_{\text{fin}}$: number of tasks completed in this cycle
- $N_{\text{fail}}$: number of tasks dropped since last cycle
- $\alpha$: reward unit per task

### 3.1.1 Sparse reward shaping term

The sparsity and delay of the reward term would make training exceedingly difficult. Hence we define a potential based reward shaping function:

$$F = \Phi(s') - \Phi(s)$$

where $\Phi(s)$ is the potential function combining several components:

#### Potential function

$$\Phi(s) = w_g\Big(T_{rt} + T_{pp} + T_{cp} + T_{ovl}\Big)$$

- $w_g$ is a global weight factor

- **Response time term**

$$T_{rt} = -\frac{w_r}{1 + RT}$$

  - $RT$ is the average response time.
  - $w_r$ is the response time term weighting factor
- **Processed fraction term**

$$T_{pp} = -w_f \cdot \frac{TTP}{1 + TTD + TTP}$$

  - $TTP$ is the total number of tasks processed
  - $TTD$ is the total number of tasks dropped
  - $w_f$ is the processed fraction term weighting factor
- **Communication penalty term**

$$T_{cp} = w_c \cdot TTO$$

  - $TTO$ is the total overloaded nodes
  - $w_c$ is the communication fraction term weighting factor
- **Overload Queue Term**

$$T_{ovl} = -w_o \cdot q_{\%}$$

  - $q_{\%}$ is the fraction of queue filled
  - $w_o$ is the overload queue term weighting factor

## 3.2 FAuNO (T-WP5-05): Federated AI network orchestrator

Since D5.2 [2], the development of FAuNO (T-WP5-05) has progressed further, including the preparation of a submission for the federated reinforcement learning-based orchestration component. This component has since been extended to operate outside PeersimGym in a project referred to as FAuNO Standalone. FAuNO Standalone is designed as a pluggable orchestration add-on to Babel (T-WP6-04).

### 3.2.1 FAuNO (T-WP5-05) Federated Reinforcement Learning (FRL) algorithm

The core design of the FAuNO (T-WP5-05) Orchestration component remains the same. To recap from D5.2, FAuNO's orchestration follows a standard Client/Server Federated setup, divided into local and global training components. Local training uses Proximal Policy Optimization (PPO) [19], an Actor-Critic RL algorithm. Both actor and critic are implemented via deep function approximators. For global training, we adapt the Federated Buffering (FedBuff) [20] approach, using a buffered asynchronous scheme where the server aggregates the first *K* updates rather than waiting for all participants. Our extension supports continuous control: agents continuously train and send updates, replacing previous ones and applying weighted aggregation. Once sufficient updates are collected, a FedAvg-style aggregation updates the global model, which is then distributed to all participants.

We continued development based on the FAuNO results, culminating in a paper currently under submission to Neural Information Processing Systems Conference (NeurIPS). The

testing workflow was streamlined, with mechanisms added to support rapid experimental iteration.

We further fine-tuned the model to improve its performance. Lastly, we implemented the state-of-the-art algorithm for task offloading in edge systems, Synchronous Cooperative Offloading Framework (SCOF) [21], and benchmarked FAuNO (T-WP5-05) against it. We make available the latest version of FAuNO (T-WP5-05) in an anonymized repository [22]. The repository is temporarily anonymized due to conference submission constraints.

#### 3.2.1.1 Performance evaluation

In this section, we evaluate FAuNO's performance using two standard Task Offloading (TO) metrics: average task completion time and the percentage of completed tasks, defined as the proportion of tasks created whose results were successfully returned to the originating client. We compare FAuNO (T-WP5-05) against two baseline algorithms: Least Queues (LQ), which offloads tasks to the worker with the shortest queue, and SCOF, a synchronous Federated Reinforcement Learning (FRL) solution adapted to our setting.

Benchmarks are executed in PeersimGym, using both realistic topologies generated by the Ether tool and synthetic artificial networks. Ether-based topologies are structured as hierarchical star networks, with resource-rich servers supporting small sets of client nodes, and a central high-capacity server interconnecting intermediate servers. Artificial topologies consist of 10 and 15 nodes randomly distributed in a 100×100 area, where the number of high-capacity nodes remains fixed while client nodes increase. All tests employ realistic workloads generated by PeersimGym's integration with the Alibaba Cluster Trace workload generator, rescaled to match the characteristics of edge devices. Each algorithm is trained for 40 episodes, corresponding to 400,000 simulation steps, with evaluation carried out during training.

The reported results are averages across the full training horizon. In addition to topological performance, we also examine FAuNO's learning under heterogeneous conditions. This includes analyzing how variations in device capacity and workload distribution affect convergence and coordination among agents. Hyperparameters, workload configurations, and topology details are provided in the repository's wiki.

#### 3.2.1.2 FAuNO performance in the Ether-based topologies

Table 5 and Table 6 present the average task completion ratio and average response time across Ether-based topologies, under different task arrival rates ($\lambda$) and topology sizes.

Table 5: Finished tasks (as a ratio of total tasks created)

| Algorithm | λ=0.5 (2) | λ=0.5 (4) | λ=1 (2) | λ=1 (4) | λ=2 (2) | λ=2 (4) |
|---|---|---|---|---|---|---|
| **FAuNO** | **0.967±0.014** | **0.957±0.008** | **0.956±0.015** | **0.957±0.010** | 0.893±0.015 | 0.896±0.012 |
| LQ | 0.943±0.004 | 0.948±0.003 | 0.943±0.004 | 0.948±0.003 | **0.910±0.005** | **0.915±0.006** |
| SCOF | 0.939±0.032 | 0.926±0.032 | 0.939±0.053 | 0.926±0.037 | 0.740±0.052 | 0.680±0.039 |

A clear trend is that increasing the number of nodes and the task arrival rate leads to performance degradation for all algorithms, primarily due to the faster exhaustion of computational resources and contention on shared resources, such as cloudlets.

Table 6: Response time (in simulation ticks)

| Algorithm | λ=0.5 (2) | λ=0.5 (4) | λ=1 (2) | λ=1 (4) | λ=2 (2) | λ=2 (4) |
|---|---|---|---|---|---|---|
| FAuNO | 148.22±12.36 | 148.82±6.43 | 175.69±12.09 | 175.86±6.42 | 215.82±8.70 | 209.29±6.28 |
| LQ | 258.41±9.07 | 239.59±6.64 | 278.21±8.92 | 256.99±7.66 | 296.60±7.79 | 269.76±5.07 |
| **SCOF** | **127.14±24.34** | **75.46±41.04** | **66.43±25.70** | **45.70±15.10** | **102.19±24.07** | **69.61±17.89** |

FAuNO (T-WP5-05) achieves the highest task completion ratio in most scenarios, maintaining robust performance as load increases. In terms of response time, FAuNO (T-WP5-05) consistently outperforms the heuristic baselines, though it is surpassed by SCOF. However, SCOF's advantage in response time comes at a steep cost in task completion, which drops significantly compared to both FAuNO (T-WP5-05) and LQ.

This trade-off is explained by node heterogeneity. SCOF applies a single global orchestration policy, which often redirects tasks away from high-capacity nodes once their queues fill. These redirected tasks tend to expire before completion, resulting in lower completion rates. Since expired tasks are excluded from the response time calculation, SCOF reports artificially low response times at the expense of actual throughput. FAuNO (T-WP5-05) avoids this issue by aligning local learning with federated aggregation, sustaining both efficiency and robustness under heterogeneous conditions.

#### 3.2.1.2 FAuNO (T-WP5-05) performance in artificial networks

As in the Ether networks, increasing the network size significantly degrades performance in both task completion rate (Table 7) and response time (Table 8). This degradation is exacerbated by maintaining a fixed number of cloudlets while increasing the number of client nodes.

Table 7: Finished tasks (as a ratio of total tasks created)

| Algorithm | λ=0.5 (10) | λ=0.5 (15) | λ=1 (10) | λ=1 (15) | λ=2 (10) | λ=2 (15) |
|---|---|---|---|---|---|---|
| FAuNO | 0.886 ± 0.036 | 0.769 ± 0.035 | 0.785 ± 0.050 | 0.596 ± 0.032 | 0.631 ± 0.035 | 0.388 ± 0.027 |
| LQ | ***0.912 ± 0.0010*** | ***0.781 ± 0.014*** | ***0.858 ± 0.019*** | ***0.654 ± 0.023*** | ***0.768 ± 0.037*** | ***0.441 ± 0.032*** |
| SCOF | 0.8720 ± 0.0360 | 0.7036 ± 0.0542 | 0.7707 ± 0.0548 | 0.5703 ± 0.0413 | 0.6129 ± 0.0608 | 0.3473 ± 0.0249 |

In random topologies, the LQ algorithm slightly outperforms FAuNO (T-WP5-05) in task completion. This stronger throughput reflects topology-specific dynamics: random topologies provide high connectivity between weaker Raspberry Pis (RPIs) and stronger Next Unit of Computing, i.e., NUCs (e.g., ~8.4 connections per RPI in the 15-node case). Such connectivity enables frequent offloading with relatively low delay, reducing congestion at individual NUCs.

The reward shaping used in the RL models was designed for Ether's star-like topologies, where excessive offloading risks latency spikes and bottlenecks. This global reward

structure introduces a bias toward local execution. LQ, being reactive and unconstrained by reward design, offloads more aggressively, achieving higher throughput but at the cost of substantially longer response times.

TABLE 8: RESPONSE TIME (IN SIMULATION TICKS)

| Algorithm | λ=0.5 (10) | λ=0.5 (15) | λ=1 (10) | λ=1 (15) | λ=2 (10) | λ=2 (15) |
|---|---|---|---|---|---|---|
| FAuNO | **301.32± 12.34** | 404.53± 7.55 | **337.66 ± 10.46** | 411.09 ± 4.79 | 353.16 ± 9.27 | 385.42 ± 4.90 |
| LQ | 377.32 ± 11.34 | 439.73 ± 8.26 | 401.64 ± 17.04 | 433.19 ± 5.82 | 408.26 ± 10.38 | 388.61 ± 4.51 |
| **SCOF** | 308.27 ± 14.55 | ***384.71 ± 19.17*** | 337.99 ± 13.87 | ***394.92 ± 22.43*** | ***349.90 ± 17.52*** | ***363.63 ± 8.93*** |

SCOF exhibits the opposite behavior, due to the presence of more powerful nodes distributed across the network compared to the Ether scenario — its centralized, non-personalized orchestration favors local processing. This reduces communication overhead and improves response time, albeit at the expense of lower task completion rates. FAuNO (T-WP5-05) is able to learn the nuances of the environment better, achieving a balanced trade-off between the two metrics. As the number of nodes grows and the proportion of weaker nodes increases, in some tests, FAuNO (T-WP5-05) even surpasses SCOF in response time while maintaining a competitive task completion rate relative to LQ.

#### 3.2.1.3 FAuNO (T-WP5-05) learning given heterogeneity

To evaluate FAuNO's stability under asynchronous, non-IID conditions, we designed a heterogeneous workload experiment to evaluate whether FAuNO's federated critic converges stably under asynchronous, non-IID conditions, the precise setting where policy inconsistency would arise. We focus on the 15-node random topology from the paper, partitioned into 3 classes, each designed to process a workload class with different task types varying in number of instructions, $\rho$ in MBytes, data size, $\alpha^{in}$ and task arrival rate $\lambda$. The workload classes are:

- **W1** ($\lambda = 0.5; \rho = 38; \alpha^{in} = 32e7$): nuc:2, rpi5 8G:2, rpi5 6G:1
- **W2** ($\lambda = 1; \rho = 16; \alpha^{in} = 64e7$): nuc:2, rpi5 8G:2, rpi4:1
- **W3** ($\lambda = 2.0; \rho = 64; \alpha^{in} = 16e7$): nuc:1, rpi5 6G:2, rpi4:2

Clients in each region generated tasks only from their corresponding class distribution. We then compared three training setups: FAuNO (T-WP5-05) with federated critic aggregation, a pure Multi-Agent Reinforcement Learning (MARL) PPO baseline where agents learn independently without a shared critic, and a centralized oracle where all nodes share a single critic model. To assess consistency between the different critics, we introduced a critic-agreement protocol. During each evaluation episode, we sampled 500 global states and collected observations from each agent.

A value prediction matrix was built for all critics, and pair-wise Root Mean Squared Error (RMSE) scores were computed. From these, we derived a global disagreement score, which quantifies how much the value estimates diverge. The complete critic-agreement protocol is explained below in more detail. To correct for the fact that agents processing faster task streams naturally accumulate higher rewards, the corresponding task arrival rate normalized all values before computing disagreement.

**Critic-agreement protocol**

1. **Evaluation set**
   For $M$ states, $\{s_m\}$, collect the observations $o_i(s_m)$ of each agent of that given state. During an evaluation episode, we sample around $M = 500$ distinct global states.

2. **Value Prediction matrix**
   We then build matrix $V \in \mathbb{R}^{N \times M}$, where line $i$ represents the evaluation of critic $i$ and column $m$ is the evaluation of point $m$. Thus, entry $V_{i,m}$ is defined as:

   $$V_{i,m} = V_i(o_i(s_m))$$

   We also build the $V_*$ matrix with the evaluation of the centralized critic model for the same observations.

3. **Pair-wise RMSE**

   We then compute our metrics. A matrix where for each agent we have the RMSE between the different state evaluations. This metric captures the differences in evaluating the observations by each of the agents. Each entry of this matrix is given by:

   $$\text{RMSE}ij = \sqrt{\frac{1}{M}\sum_m (V_{i,m} - V_{j,m})^2}$$

   Where $V_{i,m}$ is the evaluation of agent $i$. To provide a global metric of divergence of the matrix we consider the global disagreement score computed as:

   $$\delta = \frac{2}{N(N-1)} \sum_{i<j} \text{RMSE}_{ij}$$

When normalized the critic outputted State-values are normalized by the task arrival: $\tilde{V}_{i,m} = V_{i,m}/\lambda_i$.

Results are summarized in Table 9 and Table 10. As expected, disagreement between critics decreases when the packet-drop rate is reduced, indicating more consistent models as communication becomes more reliable. FAuNO's (T-WP5-05) critics approach the predictions of the centralized oracle at low drop rates, confirming that aggregation yields stable shared learning.

By contrast, the pure MARL variant showed substantially higher disagreement, highlighting its divergence under heterogeneous workloads. These results confirm that FAuNO is robust to non-IID conditions and mitigates policy inconsistency even when agents face systematically different task distributions.

Table 9: Global disagreement score (Lower=Better)

| Variant ↓ / Packet drop $D\rightarrow$ | 0.3 | 0.5 | 0.8 |
|---|---|---|---|
| FAuNO vs FAuNO | 74.5 | 232.7 | 289.8 |
| FAuNO vs Oracle critic | 63.6 | 240.1 | 307.2 |
| Pure MARL ($D = 1$) | 147.54 | 262.22 | 313.81 |

Table 10: Disagreement scores - pure MARL vs centralized oracle ($D = 1.0$)

| Variant | Global disagreement $\delta$ |
|---|---|
| Pure MARL vs Pure MARL | 319.35 |
| Fully centralized oracle vs MARL | 325.55 |

#### 3.2.1.4 *K* exploration

We further assessed FAuNO's (T-WP5-05) robustness to stragglers and different aggregation thresholds (Table 11). In this experiment, $s$ denotes the fraction of gradients dropped before aggregation. Even under extreme conditions, with 80% of gradients dropped, both the finished-task ratio and average response time remained within one standard deviation of their baseline values. This shows that FAuNO (T-WP5-05) gracefully degrades to local MARL when connectivity is limited, without collapsing performance. We also tested sensitivity to buffer size by doubling the aggregation threshold from $K = 0.3$ to $K = 0.5$. The effect on both metrics was less than 1%, confirming that FAuNO (T-WP5-05) remains stable under variations in buffer size.

Table 11: Straggler and aggregation-threshold ablation results (mean ± s.d)

| Setting | Finished-task ratio | Avg. response time (ticks) |
|---|---|---|
| K = 0.3, s = 0.3 | 0.77 ± 0.02 | 258.85 ± 5.95 |
| K = 0.3, s = 0.5 | 0.76 ± 0.02 | 259.74 ± 6.91 |
| K = 0.3, s = 0.8 | 0.76 ± 0.02 | 260.07 ± 7.02 |
| K = 0.5, s = 0.3 | 0.76 ± 0.02 | 259.09 ± 6.35 |
| K = 0.5, s = 0.5 | 0.76 ± 0.02 | 258.83 ± 7.71 |
| K = 0.5, s = 0.8 | 0.76 ± 0.02 | 260.86 ± 6.97 |

### 3.2.2 FAuNO (T-WP5-05) Standalone

We continued the development of the FAuNO (T-WP5-05) framework and we integrating FAuNO (T-WP5-05) to extend the babel framework and provide decentralized orchestration for task processing networks of nodes, we call this component of the project as FAuNO

(T-WP5-05) Standalone (FAuNO S), and the participating nodes as FAuNO (T-WP5-05) Standalone Nodes (FSn). The idea of FAuNO S is to fulfill the capabilities mentioned in the deployment diagram in Figure 5.

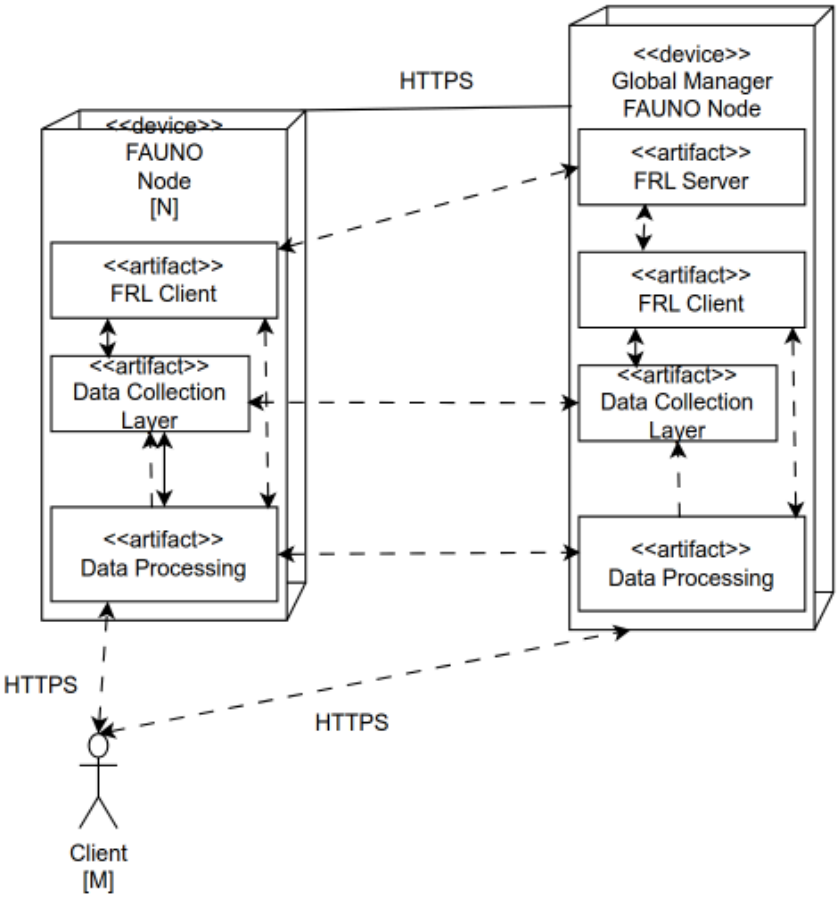


FIGURE 5: DEPLOYMENT DIAGRAM OF THE FAUNO ARCHITECTURE

### 3.2.2.1 Architecture of FAuNO (T-WP5-05) Standalone

In Figure 6, we show the full overview of the FAuNO (T-WP5-05) Standalone framework.

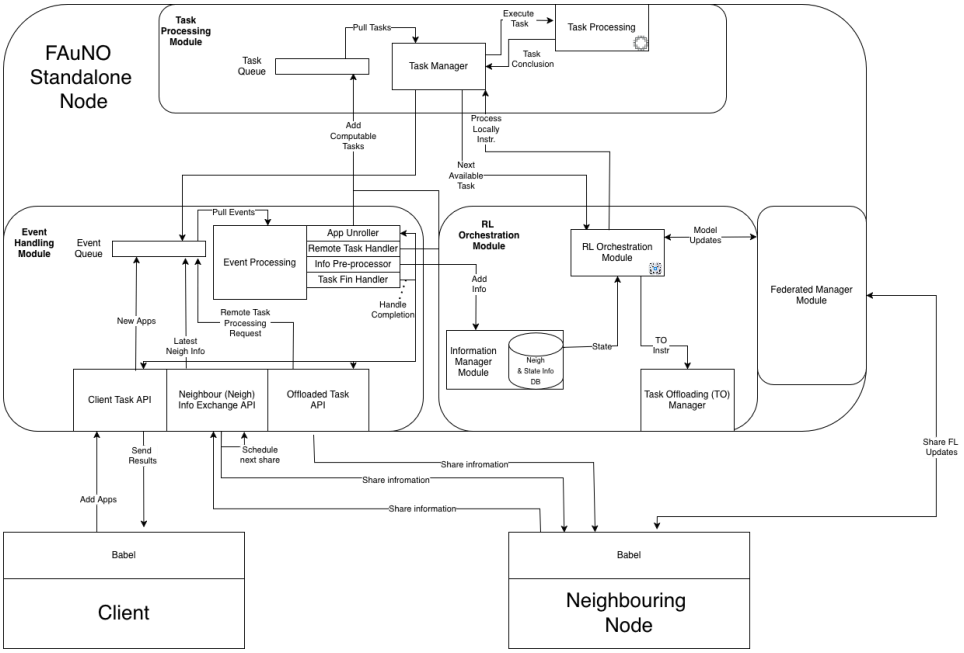


**FIGURE 6: FAUNO STANDALONE FULL FRAMEWORK OVERVIEW**

There are a total of three main modules to the FAuNO (T-WP5-05) Standalone Framework:

- Task Processing Module - Responsible for processing the tasks and management of the task queue.
- Event Handling Module - Responsible for linking the different events with the appropriate handler. There are a total of four handlers
- RL Orchestration Module + Federated Manager Module

An overview of the components is shown in Figure 6. In the subsequent sections, we will go over what each of these components entails. In addition, some modules support a special sub-component in the orchestration module that manages the federation's global model during training, we call this the global manager node.

#### *3.2.2.1.1 Task processing module*

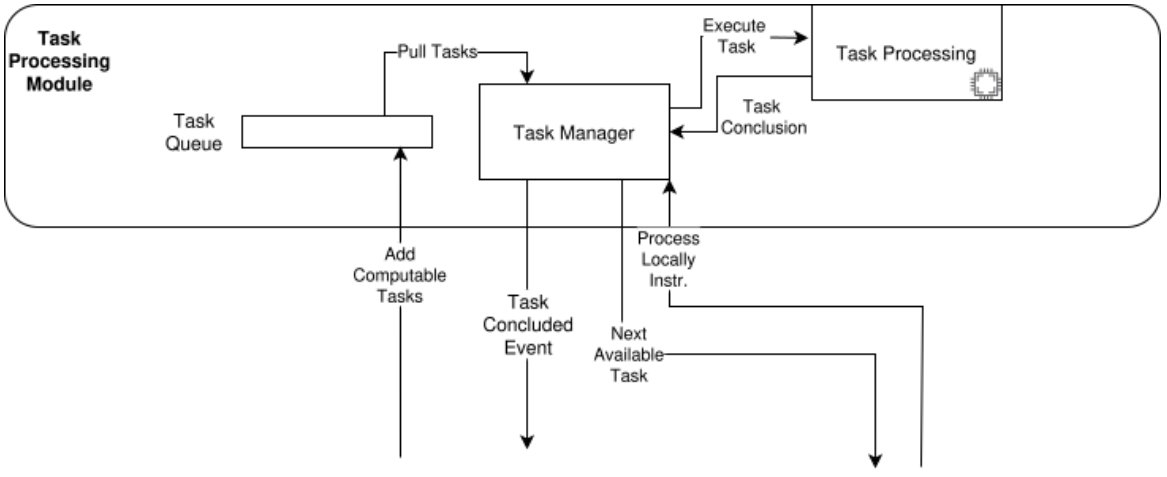


**FIGURE 7: TASK PROCESSING MODULE**

The task processing module (Figure 7) is responsible for concluding the task assigned to a given FSn, and consists of three sub-components:

- Task Queue: Stores the tasks that need to be processed. Tasks are stored in arrival order.
- Task Manager: Pulls the tasks from the queue and assigns them to a processing unit. It handles the task conclusion process and the selection of the next task to be offloaded considering that some tasks were assigned for local processing. This is the same mechanism that when prompted selects the next available task for the offloading decision
- Task Processing: The component responsible for processing the tasks. This component is still under refinement, and we are considering other approaches. Currently, task processing is handled by having clients pass executor mechanisms to the FSn, which can be used to execute any of the required processing dynamics that the client may need.

#### *3.2.2.1.2 Event managing module*

The Event managing module is shown in Figure 8.

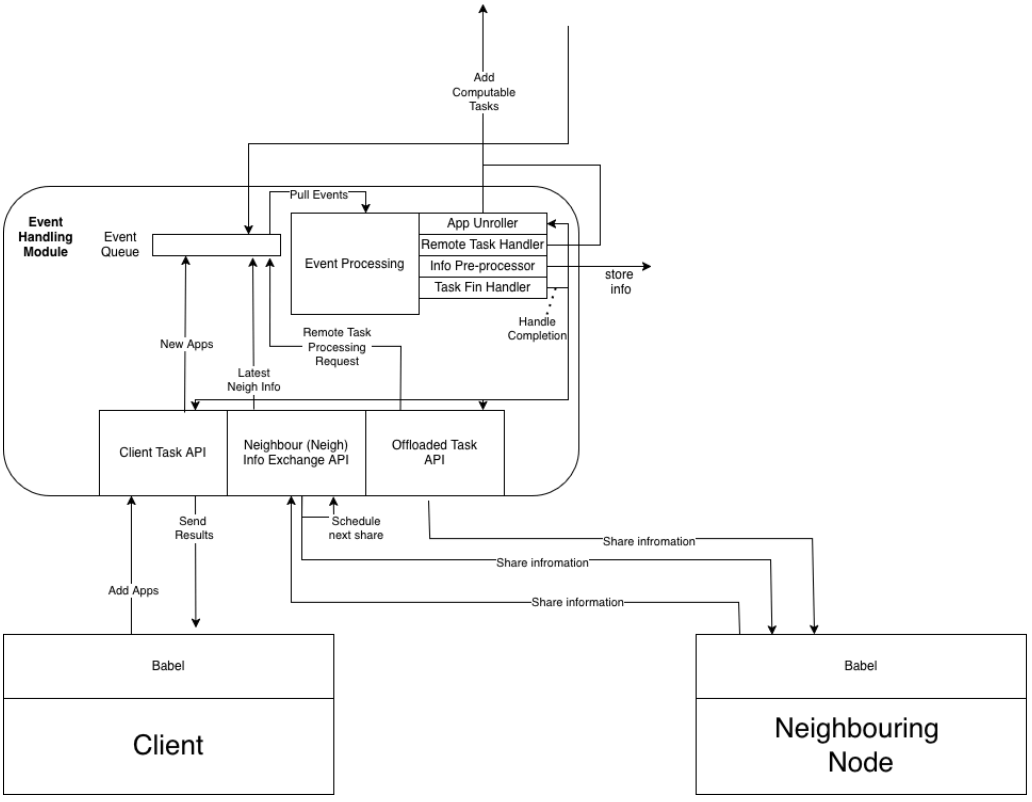


FIGURE 8: EVENT MANAGER MODULE

This component orchestrates the flow of computations. At the core of the event management component is the event queue and the event processing engine, which focuses on pulling events from the event queue and calling the appropriate processing mechanism. We can allow the addition of event-processing mechanism pairs to extend the functionality of the framework as needed. The initial predicted events are generated by the client, neighboring nodes, and the internal FSn working mechanisms.

***Client events***

We predict that there will be two types of events that involve the clients directly. The first is the generation of new applications that require processing. And the second is the sending of results from apps that reached a terminal state to the client who created the app.

1. App Generation Event: When a client creates an app in the form of a graph of tasks, these need to be registered in the node and unrolled into the executable tasks. Therefore, the event handler for the application unrolling needs to be able to maintain a data structure that tracks the progress of the apps.
2. Sending Results Event: This should be part of the joint responsibilities of a task-finishing handler and the application manager. The event that pertains to the client directly is the app conclusion, where the results are sent to the client. The other events will be elaborated on in the task events handling.

***Task events module***

The task events module manages the conversion of apps into tasks after their registration and pre-processing, and their continued unrolling into more tasks as progress is made. The main concern of this module in reality is the handling of task completions and the handling of offloaded tasks. When a task is completed one of three events is created.

1. Response of an offloaded task: An offloaded task needs to have its results sent to the node responsible for the app that generated said task.
2. Conclusion of task: Is composed of two sub-events, the updating of the progress in the app, and the addition of the remaining tasks that can be processed. If the task is completed, then the results need to be sent to the original client.
    - 2.1. Unrolling of app: Progress needs to be tracked, and the tasks unlocked by the task that concluded need to be added to the task queue.
    - 2.2. App finish: If a task concludes an app the appropriate event needs to be handled.
3. Task offloaded to node conclusion: The original node responsible for the task needs to be sent the results.
4. Receive results of an offloaded task: Have a similar handling to the local task conclusions.
5. Task Offload Event: The node prepares and shares with a neighbor one of the tasks in its task queue.

***FSn events***

The FSn events are related to communication and information exchange between nodes in the FSn network. Its main events are collecting local state information, scheduling broadcasts, and processing incoming state data from neighbors.

1. Receive Neighbor State: Handles incoming local state information from neighboring nodes, updating internal representations accordingly.

***Federated updates events***

Depending on the type of node, it may receive one or two types of FL updates. All nodes receive the latest global model. And if the node is the global model manager, it will receive the updates from all the other nodes.

1. Global Model Update: FSn receives the latest global model to update its own local version.
2. Local Model Update: The FSn responsible for managing the global model must receive the local model updates sent by the other FSns and handle them accordingly.

#### *3.2.2.1.3 Orchestration module*

The orchestration module, overviewed in Figure 9, is responsible for offloading decision-making and managing the FRL flow.

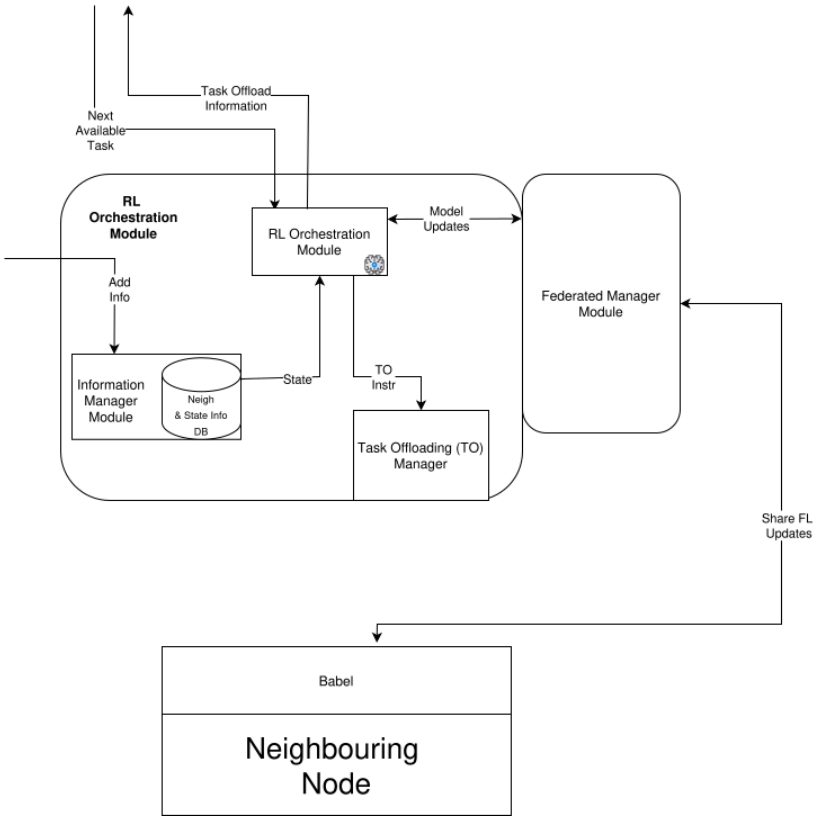


FIGURE 9: ORCHESTRATION MODULE

The Orchestration Module (OM) will continuously decide, for each task (obtained by requesting tasks to the task management module) in the queue, whether to offload it or to process it locally, based on the available information maintained by the module. If a task is to be processed locally, this information is passed to the task management module. If the task is to be offloaded an event is created and a handler that can prepare the task for offloading and send it through the appropriate Application Programming Interface (API) is called.

The OM is also responsible for handling the RL training loop, using a delayed feedback technique to acquire the necessary information for the training of the network.

The last responsibility of the OM is to handle the federated training. There are two types of nodes, the regular FSn and the global manager FSn. The regular FSn periodically after any given number of rounds of training sends the weights to the node currently managing the federation, the global manager FSn. The global manager FSn in turn is responsible for receiving the updates sent by the other nodes and storing them to aggregate into a new global model. An election mechanism in case of failure of the global manager also needs to be set in place.

#### 3.2.2.2 Integration interface with Babel

The following section describes the different types of communication requirements and their purpose in the scope of the integration with the Babel (T-WP6-04) framework. There are two types of communications: intra-node and inter-node. The interfacing with Babel works by through both Babel and the FSn supporting a Representational State Transfer (REST) interface that allows for the following communications:

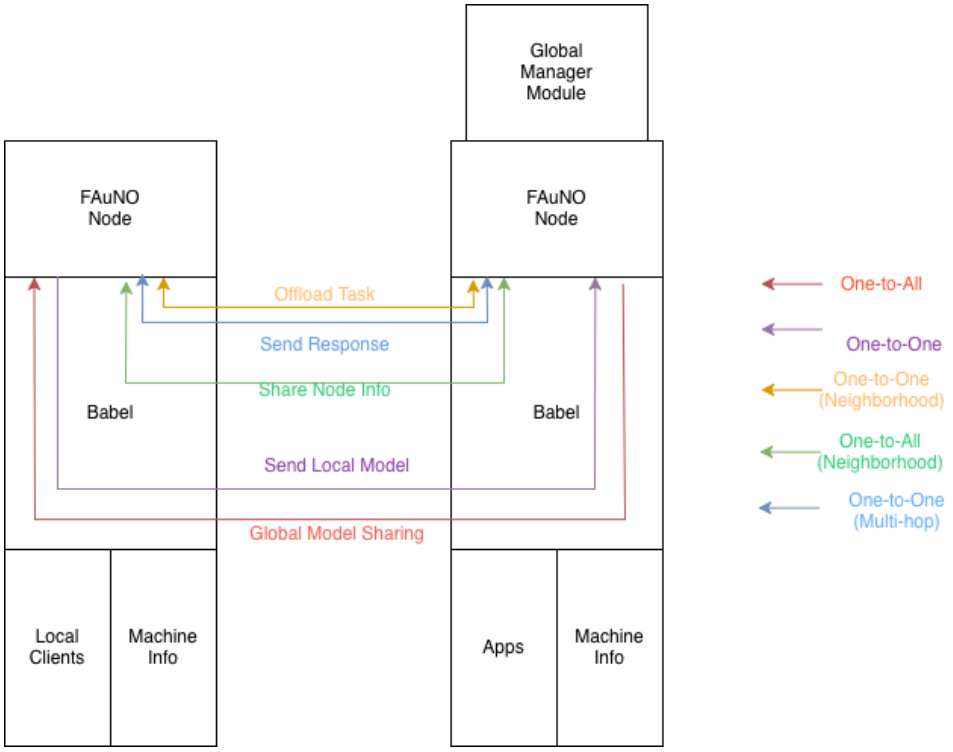


FIGURE 10: INTER-NODE COMMUNICATIONS

1. Inter-Node Communications (illustrated in Figure 10)

- Broadcast Local State Information (One-to-All): FAuNO (T-WP5-05) nodes broadcast their local state information to all neighboring nodes.
- Send Gradients to Global Manager (One-to-Any): FAuNO (T-WP5-05) nodes send gradients to the global manager node, which can be located anywhere in the network.
- Global Model Broadcast (One-to-All): The FAuNO (T-WP5-05) global manager broadcasts the updated global model to all nodes in the network.
- Task Offloading (One-to-Specific Neighbor): FAuNO (T-WP5-05) nodes offload tasks to a specific neighboring node for execution.

- Receiving Results (Multi-hop if necessary): Nodes receive the results of offloaded tasks from other nodes in the network, potentially requiring multi-hop communication if the task was offloaded through multiple nodes.

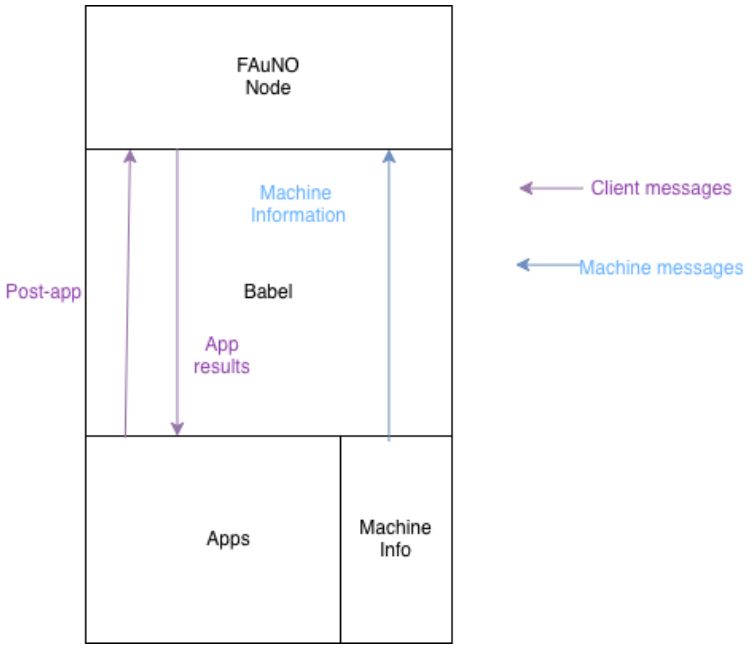


FIGURE 11: INTRA-NODE COMMUNICATION

2. Intra-Node Communications (illustrated in Figure 11)

- Clients send applications to be processed by the FSn Nodes.
- The results of the apps are sent back to the clients.
- FSn will pool Babel for new updates on the local node machine supporting information.

# 4 Advances on lightweight, energy-efficient ML techniques

In the previous deliverable D5.2 [2], we had provided development and implementation considerations regarding the lightweight ML tools that were identified in the TaRDIS architecture in D2.3 [3] and are integrated in the TaRDIS toolbox. Some of these tools have been updated during the last year of the project based on operational feedback and integration with the use cases. The tool-specific updates with reference to the D5.2 development are described below. It is worth noting that the theoretical foundation of these tools is not described here but can be traced to D5.1 [1]. Regarding the pruning (T-WP5-08), knowledge distillation (T-WP5-07) and early-exit (T-WP5-06) tools, all experimental evaluations were conducted using a representative subset of the CIFAR-10 [23] dataset, consisting of 32×32×3 Red-Green-Blue (RGB) images across 10 object classes. Moreover, the base model architecture was the VGG-16 (138M parameters, 13 convolutional layers + 3 fully connected layers), while the training configuration parameters include the SGD optimizer, a learning rate=0.001 with cosine decay, a batch_size=128 and 200 training epochs. The baseline inference performance of the model to be tested against is a 92.5% test set accuracy, while the inference latency per sample is 2.35ms. Finally, the model dimensions are: input layer 32×32×3 and fully connected layers of 4096→512→1024→10 classes.

## 4.1 Pruning (T-WP5-08)

The pruning tool (T-WP5-08) developed and updated during the first half of the project was tested on different models and datasets for a variety of pruning rates, in addition to the LSTM model for energy consumption related to the energy use case and can be found in its GitHub repository [24]. We tested the CIFAR-10 [23] dataset with a pre-trained visual geometry group (VGG) type model. To this end, the presented pruning analysis herein demonstrates the effectiveness of magnitude-based weight removal on VGG neural networks. The compression analysis, presented in Figure 12, illustrates the relationship between pruning rate and both model size reduction and test accuracy preservation.

The implemented methodology is the magnitude-based unstructured weight pruning with structured sparsity. The results show that progressive pruning from 0% to 80% achieves corresponding model size reductions, while maintaining acceptable accuracy levels. In addition, a post-training pruning with 50-epoch fine-tuning was performed (Learning Rate = 0.0001). At a 20% pruning rate, the model experiences minimal accuracy degradation of only 0.7% (from 92.5% to 91.8%), while achieving significant parameter reduction. The most aggressive 80% pruning scenario demonstrates substantial compression capabilities, reducing the model to 20% of its original size with a 9.4% accuracy reduction (to 83.1%). The dual-axis visualization clearly depicts the inverse relationship between compression efficiency and accuracy preservation, validating the expected trade-offs in neural network pruning methodologies.

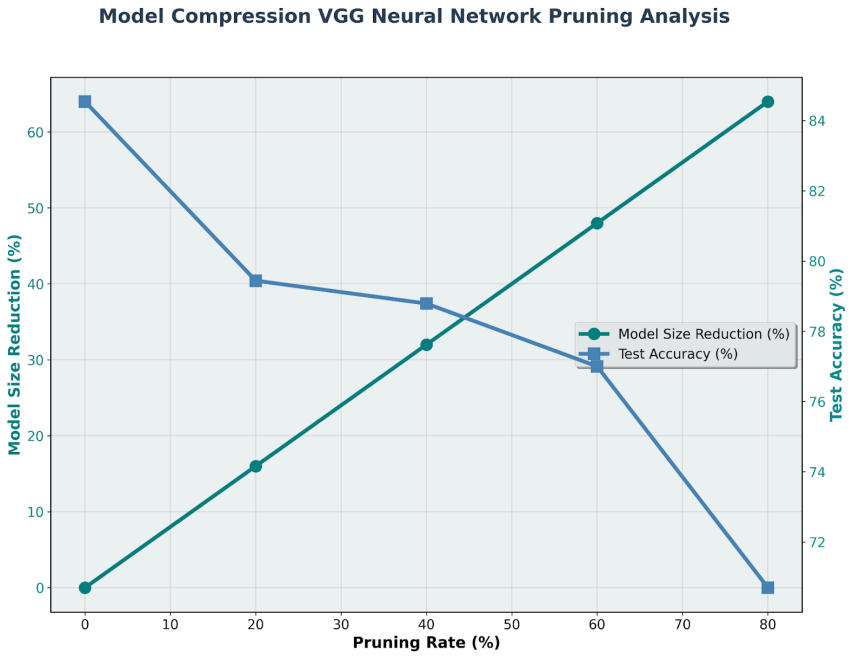


FIGURE 12: MODEL SIZE REDUCTION AND TEST ACCURACY WITH RESPECT TO PRUNING RATE

Additional analysis results are provided in Figure 13, presenting multi-metric evaluations. Specifically, the size reduction and the test accuracy are visualized with respect to the pruning rate in the top panel. Moreover, the inference time measurements and detailed compression characteristics across the full pruning spectrum are depicted in the bottom panel, showcasing the trade-off between the latency and model size reduction gains compared to the degradation of the model accuracy. Specifically, the inference latency reduction was quantified to 2.35ms→2.01ms, 1.78ms, 1.45ms, 1.12ms per sample (15%, 24%, 38%, 52% speedup).

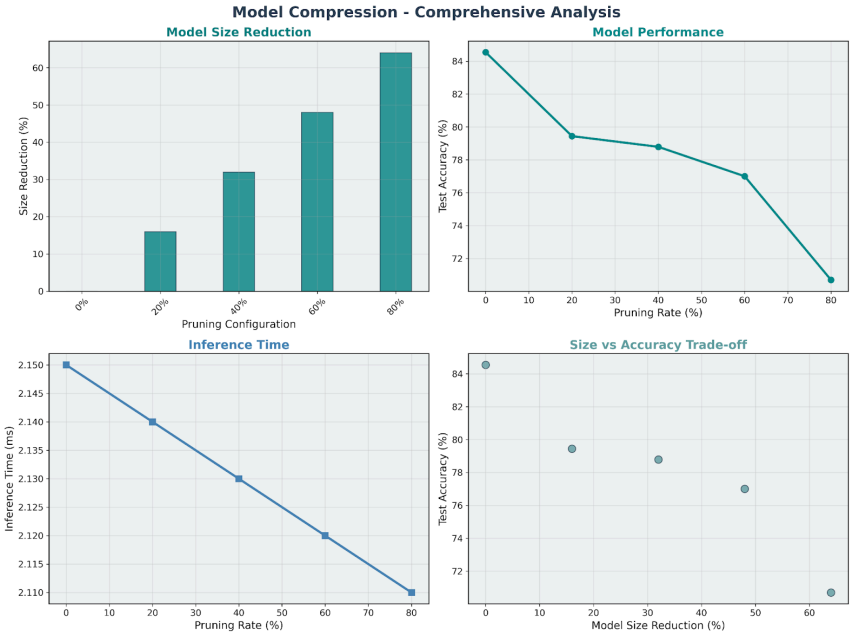


FIGURE 13: MULTI-METRIC EVALUATIONS OF THE PRUNING TOOL UTILIZING VGG NEURAL NETWORK

Furthermore, the Pruning tool (T-WP5-08) will be utilized in the EDP energy use case together with the Fedra tool. More specifically, after the finalization of the Fedra (T-WP5-09) framework and the storing of the trained models in each node, the Pruning tool (T-WP5-08) will be used to reduce the size and the inference latency of the pre-trained models hosted in the edge nodes (prosumers).

## 4.2 Early Exit of inference (T-WP5-06)

During the third year of the TaRDIS activity, the EE (T-WP5-06) tool was slightly updated from the development perspective in order to be included in the TaRDIS toolbox. Several configurations with sample datasets were tested in conjunction with the decentralized exit component. The final version of the tool can be found in its GitHub repository [25].

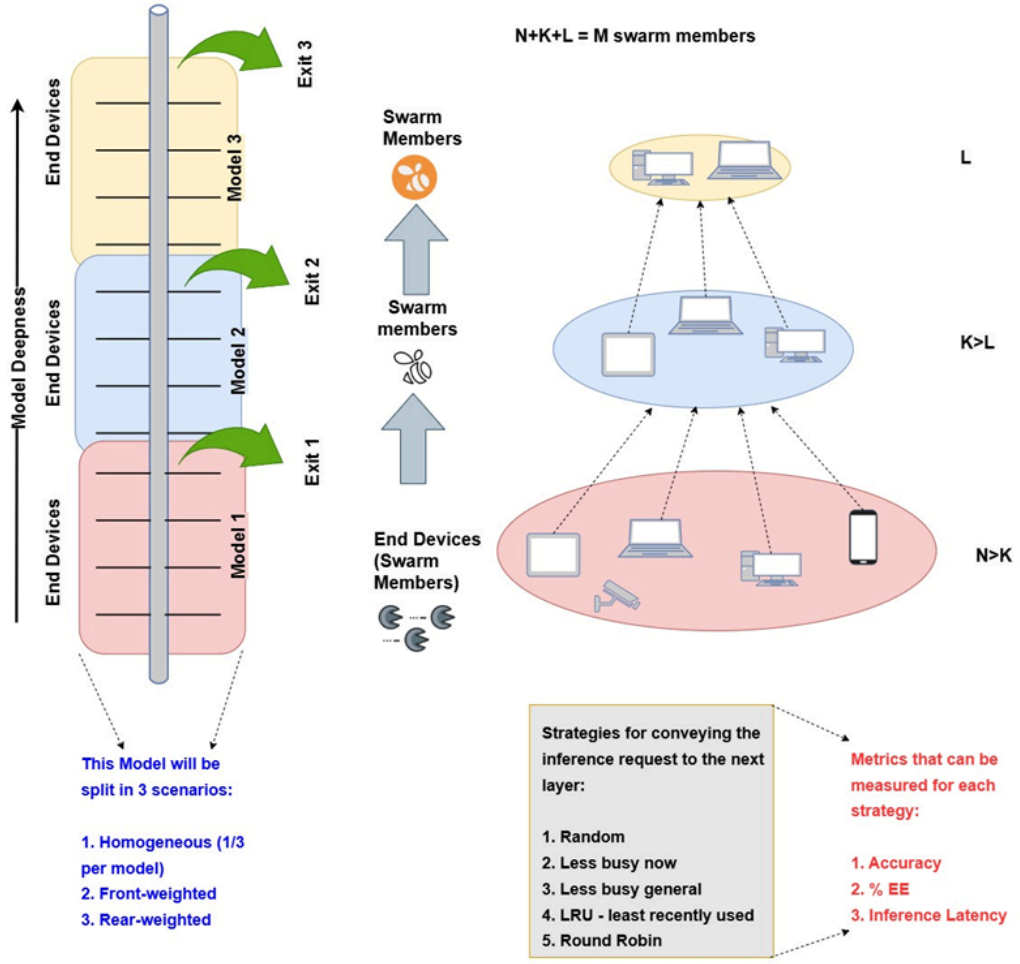


Figure 14: Different configuration options for validating the performance of the EE tool

Regarding the validation of the tool, the scenario depicted in Figure 14 is considered. Three different versions of an EE model (using the VGG model) are created, based on the splitting strategy. Specifically, the VGG-16 base architecture was utilized, enhanced with 3 intermediate classifier heads (2K, 4K, 8K parameters respectively). As an exit decision mechanism, a confidence-based thresholding was used with $\tau$=[0.9, 0.8, 0.7] for exits [0, 1, 2] and the multi-exit loss training with equal weighting across all exits was employed. The

three model variants tested were the Front-Heavy (50%/30%/20% target exit distribution), Evenly Split (33%/34%/33%), and the Rear-Heavy (35%/25%/40%).

While this subsection focuses on presenting the simulated training and testing results of the three EE models, the different configuration options will also be validated using the DEXIT tool, since the latter tool deploys the EE versions of the model in the swarm nodes in a decentralized manner, following the different strategies presented in Figure 14.

Regarding the EE tool (T-WP5-06), its implementation enables adaptive inference through intermediate classification points within the network architecture. The exit distribution analysis, displayed in Figure 15, presents the sample allocation across three strategic exit points during CIFAR-10 evaluation. Evidently, out of 10,000 test samples, the system achieved an imbalanced distribution with 3,500 samples (35%) exiting early, 2,500 samples (25%) exiting at the middle point, and 4,000 samples (40%) proceeding to the final exit for the rear-weighted EE model version. In addition, it achieves a balanced distribution with 3,300 samples exiting early, 3,400 samples exiting at the middle point, and 3,300 samples at the final exit for the evenly-split model, leading to even computational burden among the swarm participants.

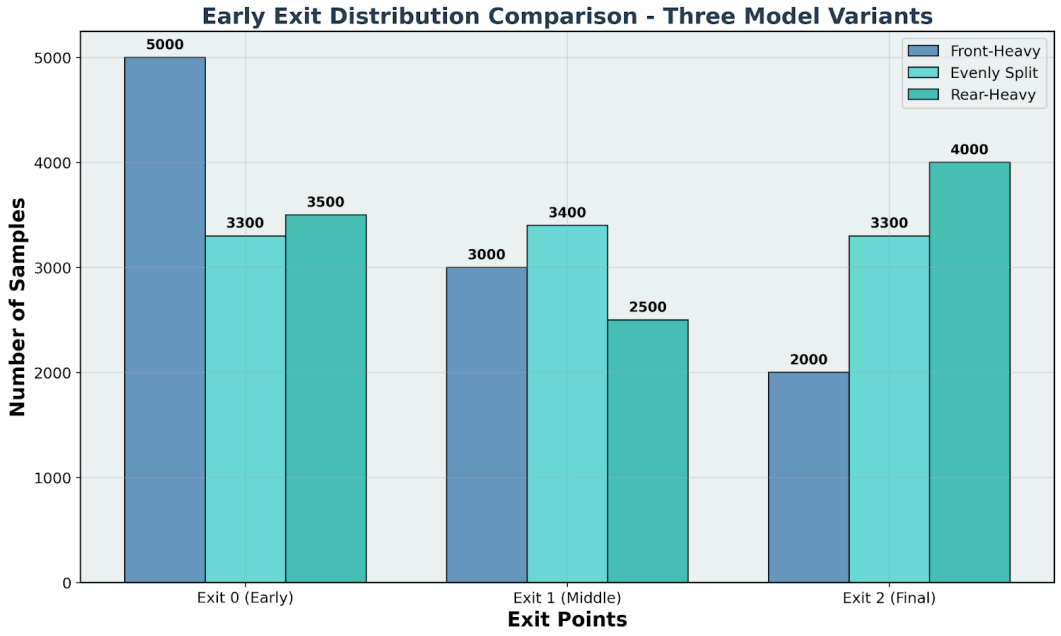


**Figure 15: Exit distribution analysis, i.e. sample allocation during the inference across the three strategic exit points for the different splits of the EE model**

Furthermore, the individual accuracies for the different exits are shown in Figure 16, where it can be seen that the testing accuracy gradually increases if the inference samples are propagated to hierarchically higher exits. Specifically, the Front-Heavy variant achieves the highest early exit accuracy (Exit 0: 81.2% vs 77.1% vs 74.8%) due to increased early computational allocation, resulting in fastest average inference (1.45ms vs 1.78ms vs 2.12ms per sample, 32-46% speedup vs 2.35ms baseline).

It should be noted that the EE models also achieve competitive overall performance while enabling significant computational efficiency gains through adaptive inference strategies. This trade-off is illustrated in Figure 17, where the overall model accuracy is plotted against the computational savings due to the early-exit during the inference process. These distributions demonstrate the system's ability to provide computational savings through

Funded by the European Union

early termination for confident predictions while maintaining the option for deeper processing via additional hidden layers when required.

The overall test accuracy decreases with early bias (84.7%, 85.2%, 86.3% respectively) but computational savings increase (45.5%, 35.0%, 33.3% average Floating Point Operation, i.e., FLOP reduction). The bar chart visualization illustrates the progressive accuracy improvement across exit points for all three architectural variants, with consistent final exit performance (92.1-92.5%) across all configurations.

Considering the validation of the Early-exit (T-WP5-06) tool along with the DEXIT tool, it will not be conducted in the framework of any of the TaRDIS use cases; however, a standalone demonstration via simulations is provided to measure the relevant metrics and KPIs associated with these tools.

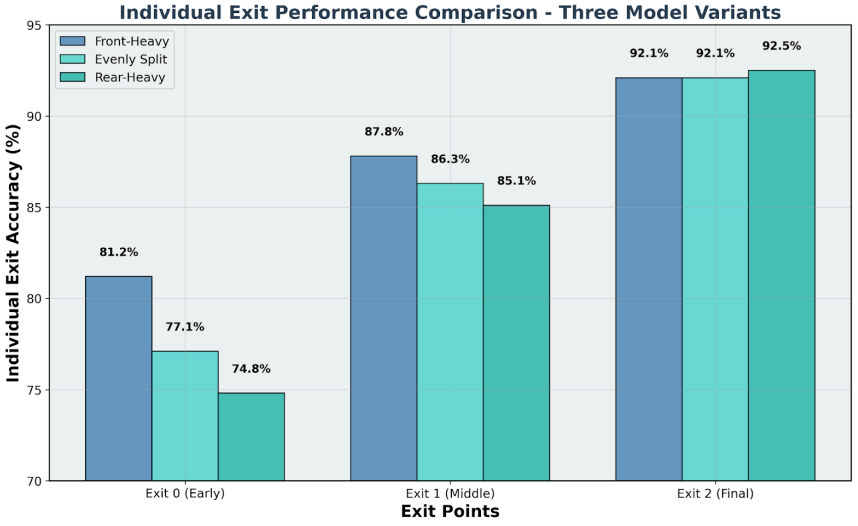


**Figure 16: Accuracy of the model for the different early exit points for the three model variants (front/heavy, evenly split, rear/heavy)**

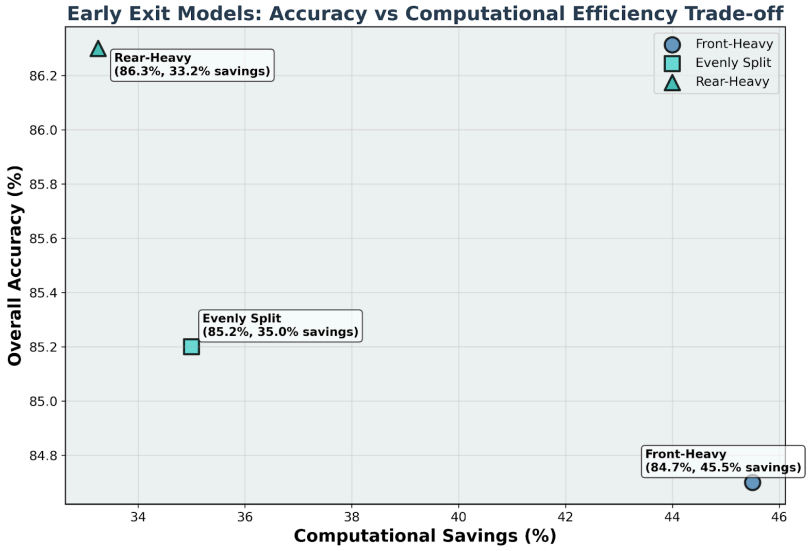


**Figure 17: Accuracy vs efficiency trade-offs across different exit configurations for the three model variants (front-heavy, evenly split, rear-heavy)**

## 4.3 Knowledge Distillation (T-WP5-07)

The KD tool (T-WP5-07) was slightly updated development-wise during the last year of the project, along with performing extensive testing using various datasets. The last version of the tool can be found in its GitHub repository [26], where an additional implementation based on Tensoflow was conducted (apart from the already existing one based on PyTorch). Concerning its evaluation, the KD (T-WP5-07) functionality demonstrates the effectiveness of teacher-student training paradigms for model compression. The CIFAR-10 dataset with the VGG model was used to validate the KD tool (T-WP5-07). The accuracy vs compression rate analysis, shown in Figure 18 compares three approaches: the baseline training (training lightweight models such as the student model from scratch), knowledge distillation (student learns from the teacher model), and the teacher model performance reference. The accuracy achieved by the teacher model is also shown as reference. The results clearly illustrate the superiority of knowledge distillation over conventional training of compressed models.

Concerning the technical details, the teacher-student training paradigm utilizes the VGG-16 teacher model (138M parameters, 92.5% test accuracy), while the student architectures include VGG-8 (29M), VGG-11 (74M), VGG-13 (97M), VGG-14 (110M), VGG-15 (124M parameters). The selected loss function was $\alpha$=0.7×CrossEntropy + 0.3×KullbackLeibler_divergence, temperature T=3.0 (optimal in the range $T \in [1.0,5.0]$). In addition, regarding the training duration KD students require 2.8-12.3 hours vs 15.2h for baseline training from scratch (19-81% training time reduction).

Specifically, at 50% compression rate, the knowledge distillation method achieved 90.1% accuracy compared to 88.7% for the baseline approach, representing a 1.4% improvement. This performance gap widened with increased compression, reaching 3.8% improvement at 95% compression (72.3% vs 68.5%). The teacher model baseline at 92.5% accuracy provides the performance ceiling, while the student model trained through the knowledge distillation approach consistently outperforms direct training of smaller architectures across the entire compression spectrum from 50% to 95%. The inference latency of the student models is 0.43-1.85ms vs 2.35ms teacher (82-43% latency reduction). The comparative plot demonstrates the superior performance of knowledge distillation over baseline training across different compression ratios, with the teacher model performance serving as the upper bound reference.

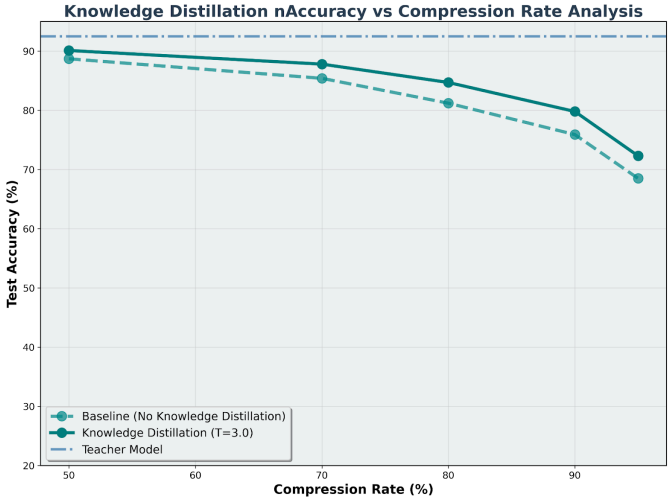


**Figure 18: Accuracy of the student model trained through the knowledge distillation method compared with the accuracy of a similar dimensionality model for various compression rates**

The temperature effects analysis, presented in Figure 19, validates the optimal temperature selection of T=3.0 for the softmax temperature function, demonstrating the importance of proper hyperparameter tuning in the knowledge distillation process.

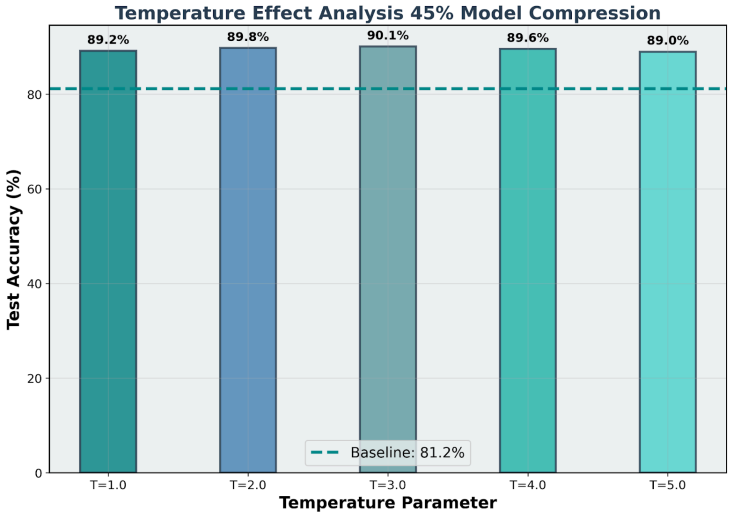


FIGURE 19: ACCURACY OF THE STUDENT MODEL FOR DIFFERENT TEMPERATURE PARAMETERS OF THE KNOWLEDGE DISTILLATION METHOD AND 45% MODEL COMPRESSION

Finally, the validation of the KD tool (T-WP5-07) will also be performed in the TID use case (federated learning in a smart home environment), where we integrated with the Federated Learning as a Service,i.e., FLaaS (T-WP5-10) tool to provide more lightweight models for inference at the edge devices, after the finalization of the FL process.

## 4.4 DEXIT FRAMEWORK

Accompanying the EE tool (T-WP5-06), the DEXIT tool was developed for enabling the deployment of the several parts of the EE models directly at the swarm nodes. The final version of the DEXIT tool can be found in its GitHub repository [27]. Since the same copy of a EE model part can be hosted in multiple swarm nodes, the architecture of Figure 14 was considered, i.e., more swarm nodes contain the initial portion of the model encompassing the first EE (denoted Internet of Things, i.e., IoT here), where less nodes host the second part of the model (denoted edge) and even less nodes (typically 1-2) host the final part of the model with exit 3 (denoted cloud). It is worth mentioning that the notation IoT-edge-cloud here may also reflect the hierarchical architectural layers of a continuum (i.e., the first part of the model is developed in the IoT device, the second model is hosted in an edge server and the last part of the model is located in the cloud), indicating that the higher layers exhibit increased computational capability than the lower-level swarm nodes.

For evaluation purposes, we produce a topology with 8 IoT, 4 edge and 2 cloud nodes, hosting the corresponding EE model splits as shown in Figure 20. It should be noted that the connection among peers follows the closest strategy, where the peers that contain a model split send their feedforward data to their closest node that contains the higher-layer model split, in case of low confidence level.

Funded by the European Union

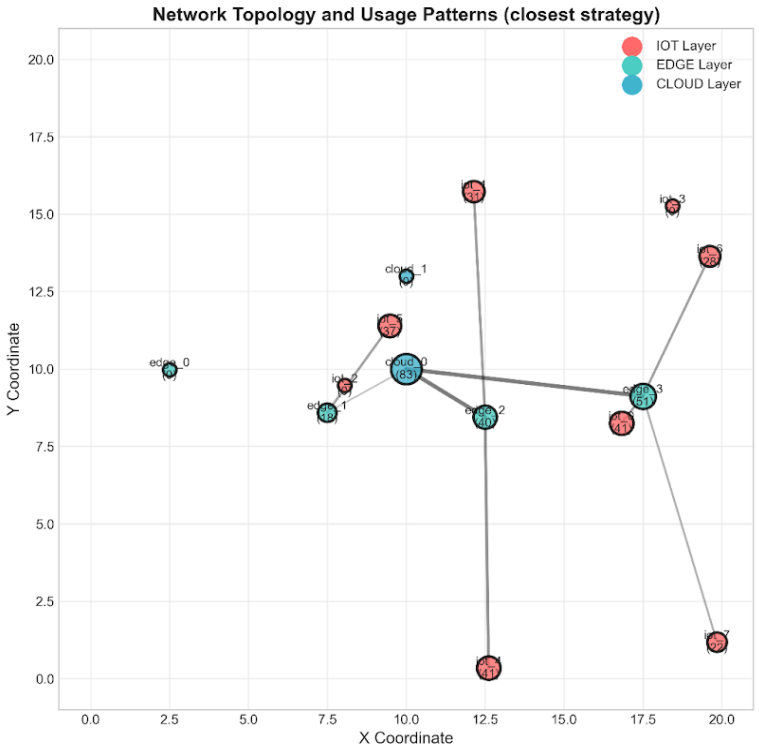


**FIGURE 20: TOPOLOGY OF THE SWARM NODES THAT HOST THE CORRESPONDING EE MODEL (IOT LAYER NODES HOST THE FIRST MODEL SPLIT, EDGE LAYER THE SECOND AND CLOUD LAYER THE LAST SPLIT)**

In this topology, we measure several metrics for the considered strategies, as shown in Figure 21, Figure 22 and Figure 23. Specifically, Figure 21 depicts the accuracy levels achieved in each exit, where the accuracy increases in higher-layer model exits, since additional hidden layers of the overall model are included in the inference process. Moreover, the confidence levels also increase in higher-tier layers, showcasing the operational levels of the confidence parameter in the practical implementation of EE inference.

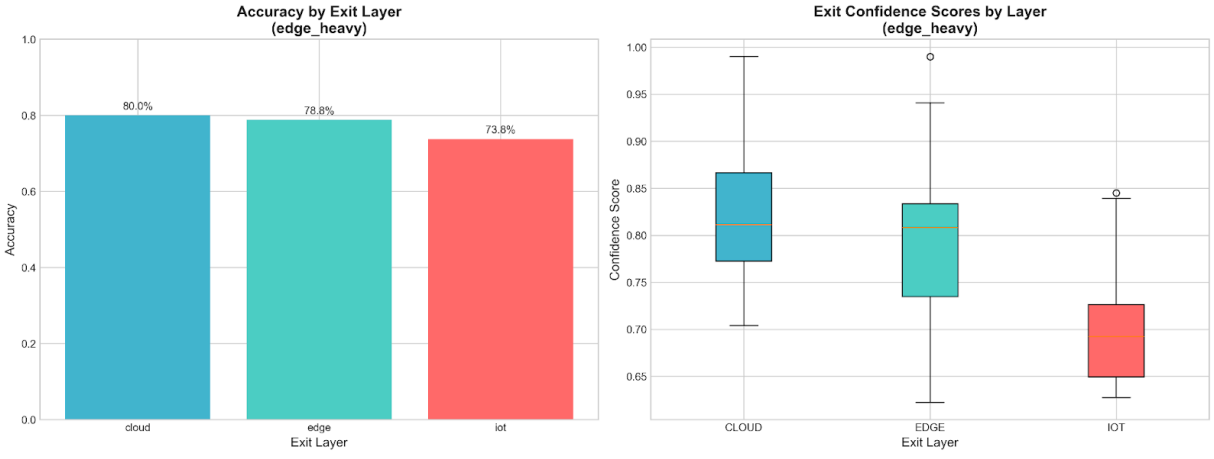


**FIGURE 21: INFERENCE ACCURACY IN EACH EXIT (LEFT) AND CONFIDENCE SCORE FOR EACH LAYER (RIGHT)**

Moreover, Figure 22 depicts the average communication overhead between the different layers/splits of the DNN model with respect to the IoT layer (zero communication overhead is present since the inference process ends in the first exit). It can be seen that the average

communication overhead is approximately 20 ms to the second model split (edge layer) and 45 ms to the third model split (cloud layer).

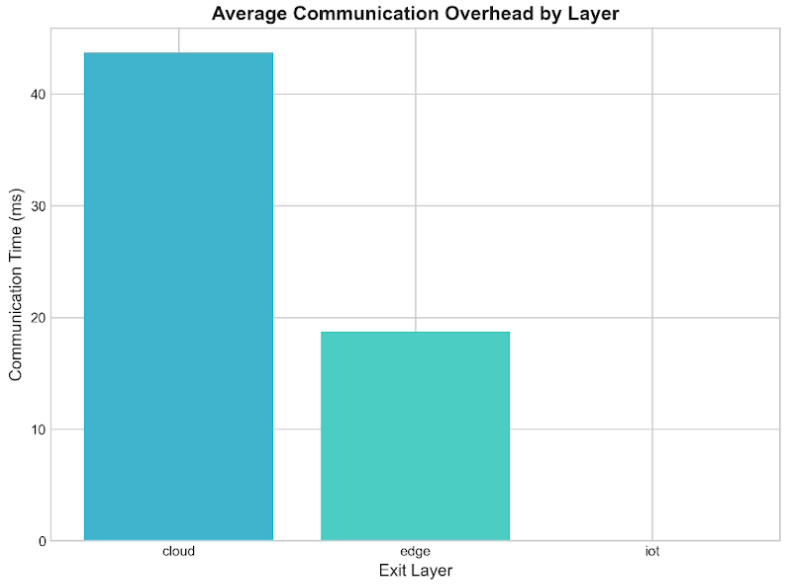


FIGURE 22: AVERAGE COMMUNICATION OVERHEAD AMONG THE HIGHER-LAYER MODEL SPLITS IN TERMS OF REQUIRED TIME

Finally, Figure 23 quantifies the CPU usage distribution for the different exits/layers, as well as the total inference time (including the model inference and the communication latency during feedforward operation). Evidently, the total inference time increases if higher-layer model splits are included in the inference process, along with the CPU usage of the participating different-layer nodes. It is worth noting that in this case, the different nodes share the computational burden of model inference compared to a full DNN model that is hosted in a single node.

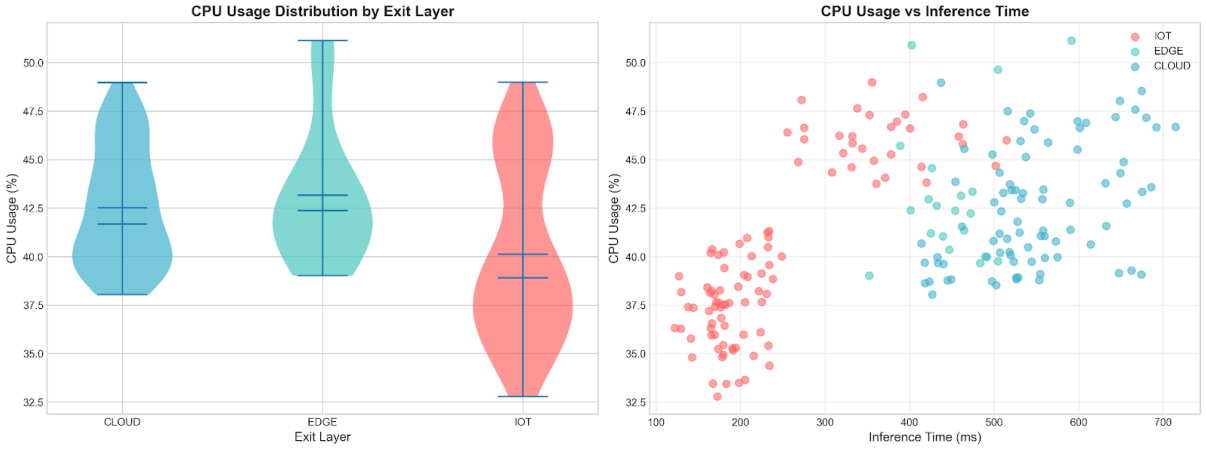


FIGURE 23: CPU USAGE DISTRIBUTION FOR THE DIFFERENT LAYERS (LEFT) AND PLOTTED AGAINST TOTAL INFERENCE TIME (RIGHT)

# 5 ML/AI TOOLS IN TARDIS

This section is dedicated to describing the roles and contributions of ML/AI tools in various aspects of TaRDIS, achieved through collaborations with other work packages.

## 5.1 ML/AI TOOLS AS BUILDING BLOCKS OF THE TARDIS TOOLBOX ARCHITECTURE, CONFORMANCE TO REQUIREMENTS

The analysis and review of the end-user needs, as well as the determination of the functional requirements of the TaRDIS development environment were carried out within WP2. In D2.2 [28], an overall requirement analysis was conducted, where WP5 contributed by defining a set of AI/ML specific requirements. D2.3 [3] was dedicated to defining a heterogeneous swarm system in the context of TaRDIS. It established the toolbox structure, including an initial view on the requirements to be fulfilled by different tools. As the development of WP5 tools advanced, we refined these relations and connected the developed tools to the corresponding requirements. In Table 12, we list these requirements, attached to the corresponding WP5 tools.

TABLE 12: AI/ML TOOLS CONFORMANCE TO WP5 REQUIREMENTS

| Requirement | Description | AI/ML tool |
|---|---|---|
| RF-WP5-FLALG-01 | A list of provided FL algorithms | - Flower-based FL model training (T-WP5-01),<br>- PTB-FLA (T-WP5-04),<br>- Fedra (T-WP5-09) |
| RF-WP5-FLALG-02 | A support for incremental model retraining within FL algorithms | - Flower-based FL model training (T-WP5-01),<br>- PTB-FLA (T-WP5-04) |
| RF-WP5-FLALG-03 | A data preprocessing facility for FL | - Data preparation for Flower-based FL model training (T-WP5-02) |
| RF-WP5-FLALG-04 | A support for ML inference and evaluation | - Flower-based FL model inference & evaluation (T-WP5-03) |
| RF-WP5-FL-ALG-05 | Support diverse ML algorithms in decentralized frameworks | - Flower-based FL model training (T-WP5-01),<br>- PTB-FLA (T-WP5-04),<br>- Fedra (T-WP5-09) |
| RF-WP5-FL-ALG-06 | Lightweight techniques for ML training and inference | - Early Exit (T-WP5-06),<br>- Knowledge Distillation (T-WP5-07),<br>- Pruning (T-WP5-08),<br>- FLaaS (T-WP5-10) |
| RF-WP5-RLALG-01 | A simulation environment for training of RL agents | - FAuNO (T-WP5-05) |
| RF-WP5-RLALG-02 | Centralized RL agent for task offloading | - FAuNO (T-WP5-05) |
| RF-WP5-RLALG-03 | Decentralized RL agent for task offloading | - FAuNO (T-WP5-05) |
| RF-WP5-GEN-01 | A list of provided FL and RL algs, including unsupervised methods, with data preprocessing, pre-trained models and incremental model retrain facilities, as well as ML inference & evaluation. | - Flower-based FL model training (T-WP5-01),<br>- PTB-FLA (T-WP5-04),<br>- Fedra (T-WP5-09) |

The requirements listed in Table 12 are met by addressing the KPIs relevant for them (see Section 7, Table 24, where we show the mapping between these requirements and KPIs, as well as the ways we address them for different tools).

The WP5 tools conforming to the use-case–specific requirements have been further refined in the context of WP7 efforts (see Section 5.5 below, where we list the connection between the AI/ML tools and the use-case requirements). These requirements are also met by addressing the corresponding KPIs.

Beside the aforementioned aspects, D2.3 [3] also introduced the TaRDIS toolbox architecture, where WP5 tools are positioned into 3 units: Intelligent orchestrator, ML inference agents and Federated ML training, as already discussed in D5.2, while the details on the architecture definition, are available in D2.3. WP5 contributed to D2.3 by providing detailed descriptions of AI/ML tools as integral parts of the toolbox as well as by recognising their connections to WP5 requirements and possible, initial conformance to use-case specific requirements.

### 5.1.1 Flower-based FL tool (T-WP5-01/02/03) compliance with requirements

As the Flower-based FL tool (T-WP5-01/02/03) consists of three TaRDIS tools, we consider them separately here, for clarity. The Flower-based FL model training tool (T-WP5-01) is required to satisfy the following WP5 requirements: RF-WP5-FLALG-01, RF-WP5-FLALG-02, RF-WP5-FLALG-05, and RF-WP5-GEN-01. The conformance with the use-case requirements will be given in D7.3 or D7.4 (where the potential use case requirements that could be of interest are the following: RNF-ACT-05, RF-ACT-13, RF-ACT-15, RF-ACT-16).

The request for KPI for the requirement RF-WP5-FLALG-01, has changed from “K-O-3.3 Reduced transmission overhead by 20% (with respect to FedAvg)” to “A list of at least 3 FL algorithms”. The Flower-based FL model training tool (T-WP5-01) satisfies this requirement, as it contains implementations of multiple FL algorithms. We already mentioned the examples before: FedAvg, Personalized Federated Learning via Moreau Envelopes (pFedMe), pFedMeNew, FedAvg with KD. The implementations are available at the Flower-based FL tool (T-WP5-01) GitHub repository [6].

Regarding RF-WP5-FLALG-02 (A support for incremental model retraining within FL algorithms), the final request is “At least one implementation”. The Flower-based FL model training tool (T-WP5-01) fulfills this request, as it offers the possibility to choose a pre-trained model and retrain it, by selecting the warm start option when starting the training, as described in Section 2.1.

The requirement RF-WP5-FLALG-05 (Support diverse ML algorithms in decentralized frameworks) is connected to the KPI “At least 3 different ML algorithms tailored to the TaRDIS use cases”. WP5 tools satisfy this requirement collectively, as they support different use cases: PTB-FLA (T-WP5-04) supports ML ODTS algorithm for the GMV use case, Fedra (T-WP5-09) is deployed at the premises of EDP and integrated with the existing infrastructure, the Flower-based FL tool (T-WP5-01/02/03) provides 2 alternative implementations for anomaly detection, which support the need described by the ACT use case, while FAuNO (T-WP5-05) is utilized in the EDP use case.

The Data preparation for Flower-based FL model training tool (T-WP5-02) directly satisfies the requirement RF-WP5-FLALG-03 (A data preprocessing facility for FL), as it provides data preparation for the Flower-based FL model training, as explained in Section 2.1.1.

Funded by the European Union

Regarding RF-WP5-GEN-01 (a list of provided FL and RL algorithms, including unsupervised methods, data preprocessing, pre-trained models, incremental model retraining facilities, as well as ML inference and evaluation), the connected KPI was updated changed to “Narrative justification”. The Flower-based FL tool (T-WP5-01/02/03) satisfies this requirement as it contains a set of implemented FL algorithms. This list also contains an unsupervised anomaly detection approach, fulfilling one aspect of this requirement. Also, the tool contains preprocessing and incremental training facilities, as well as ML inference and evaluation.

### 5.1.2 PTB-FLA and MPT-FLA (T-WP5-04) compliance with requirements

According to previous WP2 deliverables D2.2 [28] and D2.3 [3], PTB-FLA (T-WP5-04) is required to satisfy WP5 requirements RF-WP5-FLALG-01, RF-WP5-FLALG-02, RF-WP5-FLALG-05, and RF-WP5-GEN-01, as well as GMV UC requirements RF-GMV-01, RF-GMV-02, RF-GMV-11, and RF-GMV-12, RF-GMV-16, RF-GMV-17, RF-GMV-18, RF-GMV-21, RNF-GMV-01. The report on PTB-FLA (T-WP5-04) conformance with the WP5 requirements is given below, whereas the report on PTB-FLA (T-WP5-04) conformance with GMV UC requirements will be provided in D7.3 or D7.4.

Report on conformance with RF-WP5-FLALG-01 (A list of provided FL algorithms). Since the request for KPI change from “K-O-3.3 Reduced transmission overhead by 20% (wrt FedAvg)” to “A list of at least 3 FL algorithms” has been agreed on in WP5 for D5.3, PTB-FLA (T-WP5-04) satisfies this requirement, because it provides more FL algorithms, here we mention the following three: (1) the centralized logistic regression (see Section 4.1 in D5.1 [1]) – implemented as a PTB-FLA app in PTB-FLA GitHub repo [29], (2) the decentralized logistic regression – implemented as a PTB-FLA app in PTB-FLA GitHub repo [30], and (3) the centralized Modified National Institute of Standards and Technology (MNIST) (see Section 2.3.5 in D5.2) – implemented as a PTB-FLA application in PTB-FLA GitHub repo [31].

Report on conformance with RF-WP5-FLALG-02 (A support for incremental model retraining within FL algorithms). Since the request for KPI change from “N/A” to “At least one implementation” has been agreed on in WP5 for D5.3, PTB-FLA (T-WP5-04) satisfies this requirement, because the MPT-FLA application of centralized MNIST (see Section 2.3.5 in D5.2) is one such application implemented in PTB-FLA GitHub repo [32].

Report on conformance with RF-WP5-FLALG-05 (Support diverse ML algorithms in decentralized frameworks). According to the KPI for this requirement “At least 3 different ML algorithms tailored to the TaRDIS use cases”, the tools Flower-based FL (T-WP5-01/02/03), PTB-FLA (T-WP5-04), Fedra (T-WP5-09) and FAuNO (T-WP5-05) collectively satisfy this requirement as each of them supports or provides one of the ML algorithms in 3 different UCs. has been agreed on in WP5 for D5.3, In particular, PTB-FLA (T-WP5-04) supports ML ODTS algorithm for the GMV UC (see Section 6.2 in this document).

Report on conformance with RF-WP5-GEN-01 (A list of provided FL and RL algorithms, including also unsupervised methods, with data preprocessing and pre-trained models and incremental model retrain facilities, as well as ML inference and evaluation). Since the request for KPI change from “K-O-3.1, K-O-3.2” to “Narrative justification” has been agreed on in WP5 for D5.3, PTB-FLA (T-WP5-04) satisfies this requirement, because it: (1) provides a list of algorithms (see the report on conformance with RF-WP5-FLALG-01 above), (2) it provides an unsupervised algorithm (see Section 2.2.3), (3) it supports data preprocessing and pre-trained models – an example is the ML ODTS for the GMV UC, (4) incremental model retraining (see the Report on conformance with RF-WP5-FLALG-02

above), and (5) ML inference and evaluation – an example is again the ML ODTS for GMV UC (see Section 6.2).

### 5.1.3 Fedra FL tool (T-WP5-09) compliance with requirements

According to deliverables D2.2 and D2.3, the Fedra (T-WP5-09) framework is required to satisfy three WP5 requirements, namely RF-WP5-FLALG-01, RF-WP5-FL-ALG-05 and RF-WP5-GEN-01.

Report on conformance with RF-WP5-FLALG-01 (A list of provided FL algorithms). Fedra (T-WP5-09) assists in the fulfillment of this requirement, since it has been tested with multiple algorithms for anomaly detection, forecasting, as well as reinforcement learning models in the smart home energy management scenario. The final version of the Fedra tool (T-WP5-09) can be found in its GitHub repository [18]. Moreover, it is worth noting that the Fedra framework (T-WP5-09) also has a programmable component that different mechanisms of federation can be included. For instance, FedAvg, Federated Proximal Optimization (FedProx), Federated Stochastic Gradient Descent (FedSGD) are methods of fusing the local models into a global model that can be effortlessly selected in the framework.

Report on conformance with RF-WP5-FL-ALG-05 (Support diverse ML algorithms in decentralized frameworks). As mentioned in D5.2 that contains the development and implementation details of the FL framework, Fedra (T-WP5-09) is model-agnostic and thus, different types of ML models can be trained. The framework has been tested with supervised learning ML models (DNNs and LSTMs for the forecasting of the energy consumption and generation), as well as reinforcement learning models (Deep Deterministic Policy Gradient, i.e. DDPG, for the energy management system of smart homes).

Report on conformance with RF-WP5-GEN-01 (A list of provided FL and RL algorithms, including also unsupervised methods, with data preprocessing and pre-trained models and incremental model retrain facilities, as well as ML inference and evaluation). The Fedra tool (T-WP5-09) assists in the general requirement, since it has been tested with different FL mechanisms and for different types of algorithms, including the data pre-processing (in case of supervised learning algorithms), the environment configuration (in case of the reinforcement learning training playground), as well as the ML model training and inference.

### 5.1.4 FAuNO (T-WP5-05) and PeersimGym tool compliance with requirements

According to deliverables D2.2 and D2.3, the FAuNO (T-WP5-05) tool is required to meet WP5 requirements RF-WP5-RLALG-01, RF-WP5-RLALG-02 and RF-WP5-RLALG-03.

For RF-WP5-RLALG-01 (a simulation environment for RL agent training), we developed PeersimGym, which wraps the Peersim Java simulator in a Python interface compatible with MDP and Markov Game abstractions. The final version of PeersimGym is available at the GitHub repository. This enables controlled simulation of task-offloading orchestration in diverse network topologies and supports both centralized and decentralized learning.

For RF-WP5-RLALG-02 (centralized RL for task offloading), the FAuNO (T-WP5-05) repository includes a centralized training mode in which all nodes share a global critic directly. This setup resembles asynchronous actor-critic methods: nodes gather experience locally but train a common critic for policy updates directly, not having a federated update component. Decision-making remains local.

For RF-WP5-RLALG-03 (decentralized RL for task offloading), FAuNO and FAuNO Standalone (T-WP5-05) support fully decentralized operation with distributed reinforcement

learning orchestration that is trained in a federated fashion. Each node is capable of local decision making and trains its local actor-critic pair. Periodically exchanging model updates in an asynchronous manner with a manager node for aggregation. This provides autonomous coordination without requiring centralized control and meets the requirement of decentralized RL orchestration.

By combining PeersimGym with both centralized and decentralized modes of FAuNO (T-WP5-05), the framework satisfies all three WP5 RL orchestration requirements in the TaRDIS architecture.

### 5.1.5 Lightweight ML tools (T-WP5-06/07/08) compliance with requirements

According to deliverables D2.2 [28] and D2.3 [3], the lightweight ML tools satisfy one WP5 requirement, namely RF-WP5-FL-ALG-06. The linked use case requirements will be described in deliverables D7.3 and D7.4.

Report on conformance with RF-WP5-FL-ALG-06 (Lightweight techniques for ML training and inference). The developed lightweight tools fulfil this requirement, since four tools are included in the Tardis toolkit to support lightweight methods for training (FLaaS, i.e., T-WP5-10) and inference (EE, i.e., T-WP5-06, KD, i.e., T-WP5-07 and pruning, i.e., T-WP5-08) of ML models that reduce the overall computational complexity, while retaining the model accuracy and reducing the time required for model inference. The trade-offs between accuracy, computational resources and latency have been extensively described in Section 4 for the three lightweight tools for ML model inference, along with their GitHub links and in Section 6.4 for the training tool (FLaaS, i.e., T-WP5-10), along with the integration with KD (T-WP5-07) and the use of split learning in the training process. The KPIs for these tools are presented in Section 7.2, where it is shown that they achieve measurable improvements against baseline methods.

## 5.2 ML/AI tools integration into the TaRDIS toolbox IDE

TaRDIS proposes an event-driven programming model and runtime that provide high-level APIs for developers to utilize tools for communication, verification, machine learning, monitoring and reconfiguration in distributed swarm applications. The dedicated TaRDIS Integrated Development Environment (IDE) is meant to simplify programming and integration of these facilities. These directions are the main focus within WP3. The AI/ML tools are being continuously integrated into the TaRDIS toolbox IDE. The PTB-FLA toolkit (T-WP5-04) is already integrated into the IDE, where elements of PTB-FLA can be selected for application development. Additionally, local application instances can be launched through a GUI. The FAuNO tool (T-WP5-05) has also already been integrated with the IDE. This enables creating a FAuNO project, while choosing from several options. The rest of the WP5 tools are currently being integrated or are planned to be integrated into the IDE in the near future, by joint effort between WP3 and WP5.

TaRDIS aims to define and document a developer-oriented approach, based on lessons learned in implementing and evaluating the project use cases. This involves collecting “use case recipes”, that illustrate the toolkit usage in the use cases. Consequently, this approach shows how the TaRDIS usage integrates with mainstream development methodologies. This also represents a relevant WP3 endeavor.

The progress regarding the mentioned efforts was documented in prior WP3 deliverables, where WP5 actively contributed. In D3.1 [33], an initial vision on AI/ML APIs was introduced. In D3.2 [34], descriptions of the proposed WP5 candidate applications were

provided. The second revision of the TaRDIS programming models and TaRDIS toolkit APIs, in D3.3 [35], also contained updated and refined descriptions on WP5 APIs. Furthermore, in the second report on IDE, i.e. D3.4 [13], the statuses of integrating TaRDIS tools, including AI/ML tools, to the IDE were discussed. WP5 also provided and updated an overview of its APIs, under the TaRDIS wiki. Recently, D3.5 [36] has been submitted, as a third report on programming models and APIs, where WP5 contributed by providing updates about the APIs, corresponding to WP5 tools and also by participating in the internal review process. This deliverable introduces a correct by construction managed swarm application development via PTB-FLA (T-WP5-04) as one of main novelties. Currently, the "use case recipes" are still being collected, with the aim to document the TaRDIS developer approach in D3.6. WP5 actively works on the contributions to the recipes and developer approach, by continuously collaborating with the use cases.

## 5.3 Formal verification of ML/AI algorithms

TaRDIS ensures the development of formal analyses that can determine soundness, security, integrity and reliability properties of models in a heterogeneous swarm. As already reported in D5.2 [2], WP5 collaborated with WP4 through different stages of work, resulting in two joint publications. In D4.1 [37], the properties regarding the TaRDIS use cases that need analysis and verification were categorized. Also, WP5 provided some initial definitions of properties to be developed, such as FL roles of agents, FL data privacy, FL message delivery, FL clients' equality. Results on formal verification of the correctness of PTB-FLA (T-WP5-04) FL algorithms [38] were reported in D4.2 [39]. The correctness of two generic FL algorithms was verified by proving deadlock freedom and successful FL algorithm termination properties. These properties have been formalised in communicating sequential processes calculus (CSP) [40] and verified in the Process Analysis Toolkit (PAT) [41], but there is also ongoing work on a different approach, where Maude is used for verification. Also, a systematic approach is being developed to translate Python code that follows a restricted actor-based programming model to a corresponding CSP model. Additionally, WP4 was working on an extension of Multiparty Asynchronous Session Types (MPST), which is a class of behavioural types tailored for describing distributed protocols relying on asynchronous communications [42]. The extension could enable modelling algorithms introduced in [41].

## 5.4 ML/AI tools relations with data management and distribution primitives

The development of decentralized distributed communication, membership and data management primitives is the main objective of WP6. In D5.1 [1] and D5.2 [2], we described the possible usage of monitoring-related data for informed decision making. The services designed by WP6 are able to store monitoring-related data and expose it via defined APIs to other elements of the toolbox, while various WP5 tools are able to be trained on different datasets, in order to perform specific tasks. This connection can be utilized in the future, according to the specific needs. The major relations within WP5 and WP6 include integrations of AI/ML tools with the Babel framework (T-WP6-04), developed within WP6. First, a PTB-FLA (T-WP5-04) to Babel adapter was developed, in order to allow PTB-FLA based FL applications to run on multiple hosts, without requiring any changes to the code (see Section 2.2.1). The approach relies on Babel and its membership discovery and data dissemination, based on gossiping protocols, making the deployment and FL learning experimentation on various hosts simpler. Additionally, the PTB-FLA to Babel adapter has been also utilized for the GMV use case (see Section 2.2.2), adjusted slightly in order to meet the use case specific needs. The second direction regarding integration is between

the orchestrator FAuNO (T-WP5-05) and the Babel framework, where FAuNO Standalone is designed as a pluggable distributed orchestration add-on to Babel (see Section 3.2.2).

## 5.5 ML/AI TOOLS AS MEANS FOR VALIDATION AND DEMONSTRATION OF THE TaRDIS TOOLBOX

The TaRDIS technologies and methods developed through work packages 3, 4, 5 and 6 need to be aligned with the requirements identified in WP2. We already illustrated the conformance of WP5 tools to the WP5 specific requirements in Section 5.1 above. However, it is crucial to evaluate the implementations of the requirements in concrete use cases. This is the major focus of work in WP7. The use case specific requirements can be met by addressing the corresponding KPIs. These requirements and KPIs have been carefully aligned, where WP5 continuously contributed to WP7 deliverables and strived to meet use case specific requirements by WP5 tools, where possible. In Table 13, we provide an overview of WP5 tools conformance to TaRDIS use cases.

TABLE 13: WP5 TOOLS CONFORMANCE TO THE USE CASES

| WP5 tool | Use case |
|---|---|
| Flower-based FL tool (T-WP5-01/02/03) | Individual tool, tested on ACT simulated data |
| PTB-FLA and MPT-FLA (T-WP5-04) | GMV |
| Fedra (T-WP5-09) | EDP |
| FAuNO (T-WP5-05) | EDP |
| Pruning (T-WP5-08) | EDP |
| Early exit (T-WP5-06) | Individual tool |
| Knowledge Distillation (T-WP5-07) | TID |
| FLaaS (T-WP5-10) | TID |
| GMV ML model | GMV |

Table 13 illustrates the final state of the art of the usage of ML tools in TaRDIS use cases. We list all the WP5 TaRDIS architecture tools. Although FLaaS (T-WP5-10) was developed by TID, it is considered as a ML tool and we include it here for completeness. The GMV ML model is not a TaRDIS toolbox tool, as it was developed for the specific needs of the use case. However, it represents an important effort regarding use case ML model development. The tools that conform to a specific use case will address use case specific requirements through the corresponding use case specific KPIs. The details and final results on these applications will be reported within WP7. While the tools that are marked as individual tools are valuable, they did not directly fit the needs of the use cases. Now we discuss each of the individual tools separately.

The Flower-based FL tool (T-WP5-01/02/03) implemented an anomaly detection approach, inspired by the needs of the ACT use case (as already discussed in Section 2.1). The tool was tested on a simulated dataset, provided by the use case leader (see ahead Section 6.1). However, we consider it as an individual tool, as it was not implemented in a real factory setup due to the complexity of the operation process and compliance requirements constraints. The EE (T-WP5-06) and the DEXIT tools will not be implemented in any of the use cases and will be demonstrated as standalone tools via simulations, as illustrated in Section 4.4. The purpose for this standalone demonstration is that the combination of EE

(T-WP5-06) and DEXIT tools are use case-agnostic and require flexibility of the number of peers, their position, the EE model part that they host, etc. that the real use case implementations cannot provide. In this context, different topologies of the nodes and different hierarchical model layers are created in a simulated setting (where the nodes are simulated as virtual machines) in order to test several strategies of inference and measure the relevant metrics and KPIs associated with these tools (inference latency, energy efficiency, etc). Noteworthy, the simulated environment is quite realistic since the nodes host different EE model splits (IoT layer nodes host the first model split, edge layer the second and cloud layer the last split) and the connectivity between the hierarchical layers varies, representing the dynamic nature of the swarm environment.

## 5.6 Contributions of ML/AI tools to TaRDIS results and outcomes

The exploitation of the TaRDIS project's outcome and results, the standardisation initiatives monitoring and the dissemination activities are the main focus of interest of WP8. A news item titled "New Federated Learning Flower Framework-based tool Demonstration Showcases Interactive Tool and Advanced Algorithms" [43] was published during the recent reporting period, in M30. MARTEL and UNS collaborated to publish this demonstration, where the capabilities of the Flower-based FL tool (T-WP5-01/02/03) were demonstrated. It highlighted the capabilities and advances of FL applications. Over the course of the project, MARTEL and UNS published a total of four news items. Three of these earlier publications focused on the capabilities of the PTB-FLA tool (T-WP5-04): "ChatGPT Helps Humans Creating Federated Learning Apps on PTB-FLA" [44] in M18, "The PTB-FLA Successor MPT-FLA Advances to Edge Systems" [45] also in M18, and "PTB-FLA-Babel Stack gets rolled-out to the public: PTB-FLA got the plug-in adapter for Babel" [46], in M23. Also, WP5 published a poster for the European Telecommunications Standards Institute (ETSI) Artificial Intelligence Conference [47], as reported in D5.2 [2]. The partners involved in WP5 produced a wide range of scientific publications during the course of WP5. In Table 14 below, we provide an overview of the number of publications per task, for the overall period of the work package span. These publications are either related to the tools from WP5 tasks or represent methodological advances that are of interest for the corresponding task.

Table 14: Number of WP5 publications

| WP5 task | WP5 tool | Conference papers | Journals |
|---|---|---|---|
| **T5.1** Framework supporting AI/ML programming primitives | PTB-FLA & MPT-FLA (T-WP5-04) | 8 | 2 |
| **T5.1** Framework supporting AI/ML programming primitives | Flower-based FL tool (T-WP5-01/02/03) | 3 | 2 |
| **T5.2** AI-driven planning, deployment & orchestration framework | FAuNO & PeersimGym (T-WP5-05) | 1 | 0 |
| **T5.3** Library of lightweight and energy-efficient ML techniques | Pruning (T-WP5-08), KD (T-WP5-07), EE (T-WP5-06) | 3[1] | 5[2] |

[1] 1 conference paper has been submitted at the present deliverable submission time
[2] 1 of the journal papers is under review at the present deliverable submission time

The WP5 tools are described on the corresponding TaRDIS wiki documentation page [48], and the program code is available in corresponding repositories: Flower-based FL tool (T-WP5-01/02/03) [6], PTB-FLA (T-WP5-04) [49], Fedra (T-WP5-09) [18], FAuNO (T-WP5-05) [22], Pruning (T-WP5-08) [24], EE (T-WP5-06) [25], KD (T-WP5-07) [26].

# 6 ML MODELLING OF TARDIS USE CASES

In this section, we provide an overview on the final state of the art regarding the ML modelling of all four TaRDIS use cases.

## 6.1 ACT USE CASE

As already reported in previous deliverables (D5.1 [1] and D5.2 [2]), we developed a model for the anomaly detection task, inspired by the ACT use case. In D5.1 [1], we demonstrated the conceptual ML model, while in D5.2 [2], we presented the anomaly detection approach that utilizes the idea behind the HUNOD [11] method. We introduced and tested the centralized version of the algorithm on open data, using the MetroPT-3 [12] dataset. Now, we developed a fully decentralized FL version of this approach (see Section 2.1). Additionally, we also developed a transformer-based anomaly detection approach, as described in Section 2.1.

As mentioned earlier, the Flower-based FL tool (T-WP5-01/02/03) will not be implemented at real factory premises regarding the ACT use case, due to the complex operation process, involving different constraints. For this reason, the tool is not an integral part of the use case. Instead, it is considered as an individual tool, evaluated on use case specific data. The developed anomaly detection approach can still provide valuable insights into factory processes. The use case leader provided a simulated dataset for the purposes of testing the developed approaches. The simulated dataset closely approximates the characteristics of the real dataset. Therefore, the Flower-based FL tool (T-WP5-01/02/03) can be easily utilized on real, production data.

In order to enable the FL model training on the provided dataset, we used the data preparation tool to transform the data to a suitable form. We now describe the steps applied to prepare the data. The dataset was a JSON log of ACT events produced during a simulated industrial process run. Each JSON entry represents a single event, generated by a node, i.e., agent, that can be a machine, station, storage unit, etc. Each event contains some metadata, such as timestamps, tags, IDs, and content, that differs for various stream types. Basically, a multivariate time-series of events from multiple emitters (nodes) is provided.

First, we used the data preparation tool to parse and flatten the data and convert it to a tabular form, i.e., a pandas [50] DataFrame. Next, we applied timestamp normalization and feature extraction. Here we converted categorical fields to encoded values and performed normalization. We also split the data using the preprocessing tool, as the raw data was provided as a single file. We created 17 data chunks, for 17 clients, as there were logs from 17 actors in the file, i.e. 17 unique streams for ACT nodes. Each node emits approximately 40 events, resulting in 682 events in total. We split the data for each client in the following way: 70% for training, 15% for validation and 15% for testing. We run the tests on our cluster environment. This way, we can demonstrate a realistic setting, where the run starts with a lower number of clients and client participation progressively increases, as the clients connect asynchronously in the cluster.

We tested both anomaly detection approaches on this dataset. First, we discuss the results for the autoencoder-based anomaly detection approach. We observed that the training and evaluation losses decreased steadily. Since the dataset was unlabeled, we could not apply traditional accuracy metrics that compare the outcome to ground truth. However, we applied a proxy metric computation, so that we obtain a fraction of samples whose reconstruction error falls below a threshold value. During federated training, the model maintained a

Funded by the European Union

pseudo-accuracy of approximately 89.9%. This means that roughly 90% of the validation samples were reconstructed within the learned threshold, which indicates that the model is a stable representation.

In Figure 24, we can see that the threshold decreases steadily over the training rounds. This plot illustrates how the decision boundary of the model evolves during training. Here, the decrease over rounds, but the difference in actual values is small. The fact that the threshold remains almost constant across the rounds indicates that the algorithm stabilizes quickly. The absence of fluctuations in the threshold curve suggests that the model does not encounter samples that strongly deviate from the normal distribution. This confirms the assumption that this dataset does not contain anomalies, i.e., represents a healthy baseline process.

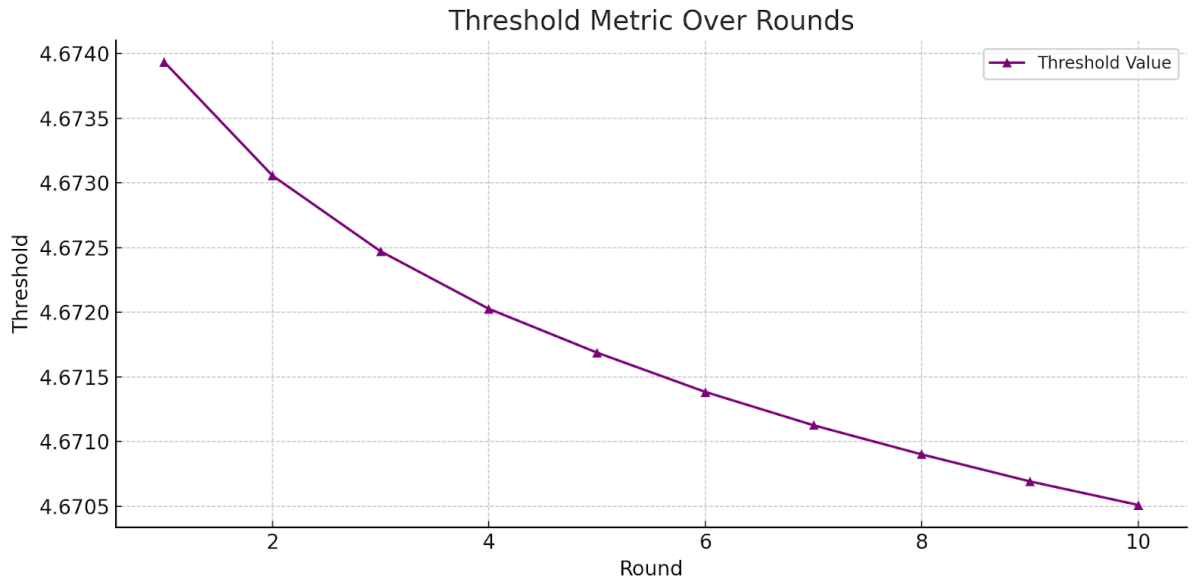


**Figure 24: Threshold metric over rounds for the autoencoder, on simulated data**

As shown in Figure 25, both training and evaluation losses remain nearly constant across rounds. This behavior arises because the dataset used in this simulation was synthetically generated from a single, stable process run without anomalies or temporal variation. Consequently, the model is exposed to highly consistent input-output relationships, leading to immediate convergence and negligible gradient updates after the initial rounds. The constant loss, therefore, reflects an already optimized and stable model rather than a failure to learn.

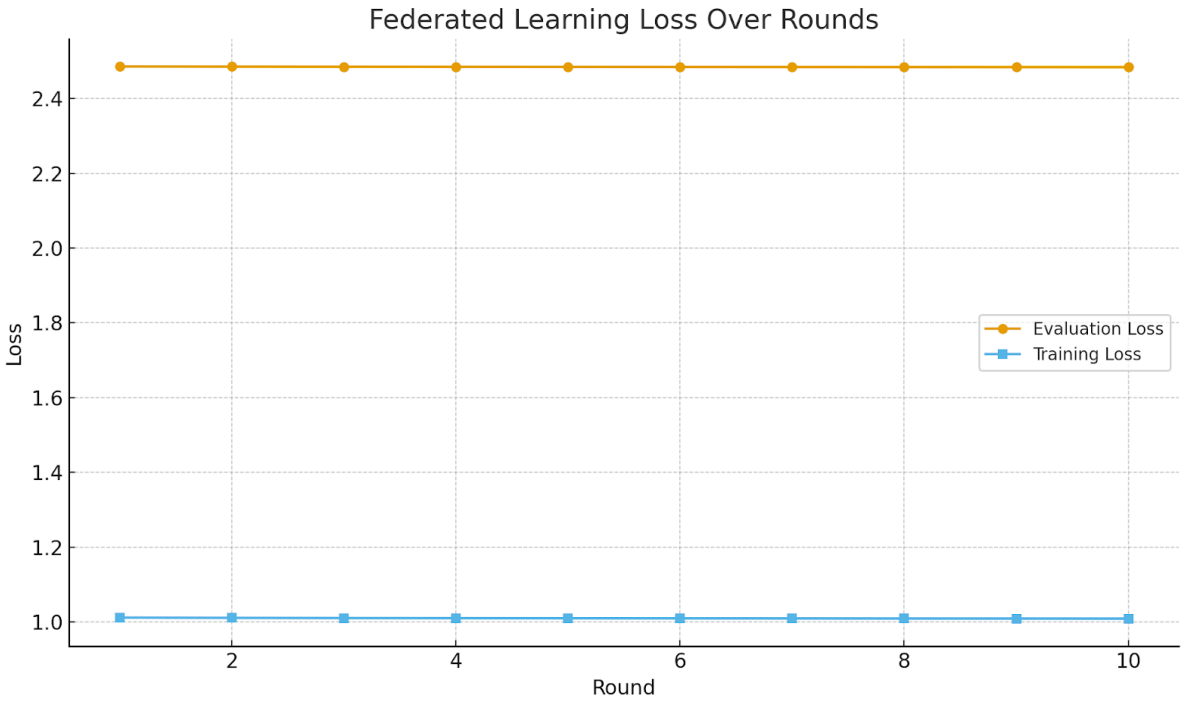


FIGURE 25: LEARNING LOSS OVER ROUNDS FOR THE AUTOENCODER, ON SIMULATED DATA

In order to make the anomaly detection process more realistic, we used the data-preparation tool to add a set of synthetic anomalies. A small random portion of the dataset is intentionally modified to simulate irregular or faulty behaviour. These changes alter numerical values, timing relationships and categorical entries. The aim is to create realistic deviations from normal patterns. We use a noise magnitude value of 0.5, meaning adding random Gaussian noise with a standard deviation of 0.5, which is subtle but noticeable. We use a shift magnitude value of ±3σ, meaning that a random subset of samples have their feature values shifted by 3 standard deviations away from the normal distribution, which is a fairly strong anomaly. These values ensure that anomalies are detectable but not trivial.

Each altered record is flagged with an indicator that marks it anomalous. We perform this step, before splitting the data to clients. In total, we make 5% of the overall dataset anomalous. However, each client may have a various percentage of anomalous samples, and also, some clients may not have anomalous samples at all. We train the clients on non anomalous data and put all the anomalies to test and validation subsets. Table 15 shows this distribution.

TABLE 15: THE DATA DISTRIBUTION WITH ANOMALIES

| Phase | Normal samples | Anomalies | Anomalies % | Usage |
|---|---|---|---|---|
| Train | 447 | 0 | 0% | Learn baseline “healthy” patterns |
| Validate | 96 | 12 | 11.1% | Estimate threshold |
| Test | 104 | 22 | 17.5% | Test anomaly detection |
| **Validation + test** | 200 | 34 | 14.5% | Mirror real-world evaluation scenario |

We now analyse the results of training and testing the model on this data. We use our cluster environment as before. We have 17 data chunks for 17 clients, but we use 15 physical nodes that are available. Instead of redistributing the data to 14 clients or creating multiple clients on one node, we will use the 17 client setup, in order to simulate a real situation, where some clients may be unavailable at the moment. The cluster makes exactly a scenario like this. The server starts, 14 clients start, and 3 clients wait in the queue, until some resources get free. We show the results for this setup in Table 16.

TABLE 16: RESULTS FOR THE AUTOENCODER, ON DATA CONTAINING ANOMALIES

| Metric | Value | Meaning |
|---|---|---|
| Average evaluation loss | 2.486 | Stable convergence |
| Average pseudo accuracy | 0.9005 | ~90% normal correctly recognized |
| Average anomaly rate | 0.0995 | ~10% flagged as anomalies |
| Average threshold | 4.68 | Tightening boundary as training progresses |

The changes in average loss and threshold are again subtle. The detection rate of 10% closely matches the injected anomaly rate of 14%. The described mechanism for adding anomalies is expected to result in 10-15% of anomalies flagged, which matches the obtained results.

We now test the transformer-based anomaly detection approach on the same data. We use the dataset with the added anomalies, to make the results more meaningful. The results are presented in Table 17.

TABLE 17: RESULTS FOR THE TRANSFORMER MODEL, ON DATA CONTAINING ANOMALIES

| Metric | Final value |
|---|---|
| Accuracy | 0.985 |
| Precision | 0.96 |
| Recall | 0.99 |
| F1-score | 0.972 |
| Loss | 2.7 |

As Table 17 shows, the model demonstrates strong classification performance across all clients. The model achieves high accuracy and precision, supported by a stable F1-score. The reconstruction losses appear slightly higher than with the autoencoder, but the transformer model captures temporal dependencies more effectively. Naturally, the transformer model achieves these results at the cost of longer per-round computation time. Nevertheless, it is suitable for sequence-aware anomaly detection with high accuracy in a federated setting.

## 6.2 GMV USE CASE

As reported in D5.2 [2], we developed an Orbit Determination algorithm using ML. As previously mentioned, the key development included in this work is integrating Physics-Informed Neural Networks (PINNs).

To do so, this model is enhanced with a gravitational layer that calculates the acceleration on the satellite, including the J2 perturbation, which is representative of the oblateness of the Earth, and performs 4 steps of numerical approximation.

### 6.2.1 Two-body dynamics layer

This layer is similar to the process of a simple propagator, but it is used inside the Neural network to expand the number of input features.
The two-body gravitational acceleration is first computed as:

$$\mathbf{a}_{\text{grav}}(x_k) = -\mu \frac{\mathbf{r}_k}{\|\mathbf{r}_k\|^3}$$

with μ the Earth's gravitational parameter.
To include the J2 perturbation, we add the oblateness effect:

$$a_{J2,x} = -\frac{3}{2} \frac{J_2 \mu R_E^2}{\|\mathbf{r}_k\|^5} \left(1 - 5\frac{z_k^2}{\|\mathbf{r}_k\|^2}\right) x_k,$$

$$a_{J2,y} = -\frac{3}{2} \frac{J_2 \mu R_E^2}{\|\mathbf{r}_k\|^5} \left(1 - 5\frac{z_k^2}{\|\mathbf{r}_k\|^2}\right) y_k,$$

$$a_{J2,z} = -\frac{3}{2} \frac{J_2 \mu R_E^2}{\|\mathbf{r}_k\|^5} \left(3 - 5\frac{z_k^2}{\|\mathbf{r}_k\|^2}\right) z_k,$$

where $J_2$ is the Earth's oblateness coefficient, $R_E$ is the Earth's equatorial radius, and [x,y,z] are the Cartesian position components.

The total acceleration is the sum of the gravitational acceleration + the J2 perturbation, and to further improve the precision of the acceleration attained by this layer, we use a Runge-Kutta integration scheme.

The output of this block is a good initial approximation to the desired result, and thus the model focus is on adapting the estimated state to the particular characteristics of the satellite and the environment.

#### 6.2.1.1 Results

In the paper "Deep Extended Kalman Filter for State Estimation for Satellite Constellations" [51], developed as part of this use case, the model is evaluated in terms of computational efficiency and precision. The paper presents a deep extended kalman filter, more precisely, a kalman filter where the "predict step" is replaced by a neural network.

The result in Table 18 evaluates the quality of the orbital determination model, when compared with a simple J2 baseline. Errors are reported as Absolute error values over 2 hours propagation. Computational cost is per 720 steps / number of satellites in the constellation.

TABLE 18: COMPARISON OF THE PROPOSED PINN, WITH A BASELINE ORBIT PROPAGATOR

| Dataset | Propagation | Position [km] | Vel. [km/s] | Time [s] |
|---|---|---|---|---|
| Train | 447 | 13.347 | 0.01518 | 0.197 |
| Validate | 96 | 12.01 | 0.01225 | 0.152 |
| Test | 104 | 5.087 | 0.00453 | 0.198 |
| **Validation + test** | 200 | 1.411 | 0.00128 | 0.151 |

This result in Table 18 shows that, when calculating the orbit propagation of satellites for a period of 2 hours, which is slightly longer than an orbital period, the DNN is able to improve upon a simple Baseline. It also showcases that, when calculating an entire set of satellites simultaneously (99 and 97 satellites respectively, for each constellation), the DNN is faster.

This result is important, as it showcases that the DNN model itself is robust to recursive propagation, that is, when the output of the model is fed to the model. However, for the use case in question, the propagation occurs within an Deep Extended Kalman Filter (DEKF) framework that combines the state estimated by the neural network with the state estimated from measurements.

First, we present how the Neural Network is integrated as part of the DEKF. Note that the integration of the NN propagator is the same both for the centralized, and the Decentralized Deep EKF (DDEKF).

In Figure 26, we show an Extended Kalman Filter (EKF) diagram. The component shown in red is necessarily replaced by a neural network, whereas the matrices shown in yellow can be computed by the neural network, though this is not mandatory.

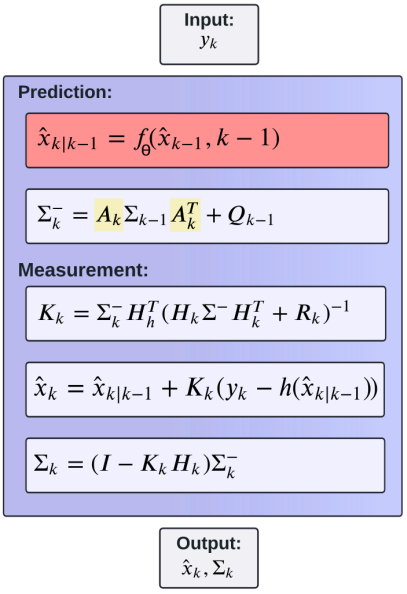


FIGURE 26: EKF DIAGRAM

#### 6.2.1.2 Preliminary results

In this section, we present some preliminary results for the neural network state predictor, evaluating the next step prediction. Since it is used as part of the DEKF, this element is the most critical component. The results presented here are preliminary since they use only a fraction of the complete dataset (around 100M individual samples) for training and testing. After testing and performing hyperparameter tuning, the final model will be trained on the complete training dataset.

The two main difficulties with this task is to minimize the bias drift that naturally occurs when the testing scenario is from a time further away from the training dataset. And preventing near-zero gradient magnitudes, as this high-precision task inherently produces very small errors, leading to correspondingly small gradient updates that can impede effective learning.

To find parameters that both obtained the best results, and reduced overfitting and bias drift, hyperparameter tuning was performed over these parameters, using 400 training epochs of the Tree-structured Parzen Estimator (TPE) sampler. Table 19 lists these values.

TABLE 19: HYPERPARAMETERS FOUND AFTER HYPERPARAMETER OPTIMIZATION

| parameters | value |
|---|---|
| hidden dimensions | 447 |
| number of layers | 2 |
| activation function | Gaussian error linear unit (Gelu) |
| weight normalization | False |
| learning rate | 6.11293e-04 |
| optimizer | AdamW[3] |

The overall results in kilometres are shown in Table 20 .

TABLE 20: RESULTS IN THE TEST SET FOR THE HYPERPARAMETER MODEL WITH A LIMITED TRAINING SET

| metric | | value |
|---|---|---|
| MAE[4] | | 0.0019 |
| RMSE | | 0.0024 |
| MAE | position [km] | 0.0019 |
| RMSE | | 0.0026 |
| MAE | velocity [km/s] | 0.0017 |
| RMSE | | 0.0021 |
| inference cost | [s/batch] | 0.005 |

The results indicate that, when predicting 10 seconds ahead, the mean average error is around 2.6 meters. We also want to guarantee that this result is not optimizing on a given

[3] Adam with Weight Decay (optimizer variant)
[4] Mean Absolute Error

dataset, and Figure 27 is a strong indication that there is no overfit, as we see that, as the epochs progress, the training, validation and test set achieve similar results. This occurs in part due to the simplicity of the model itself.

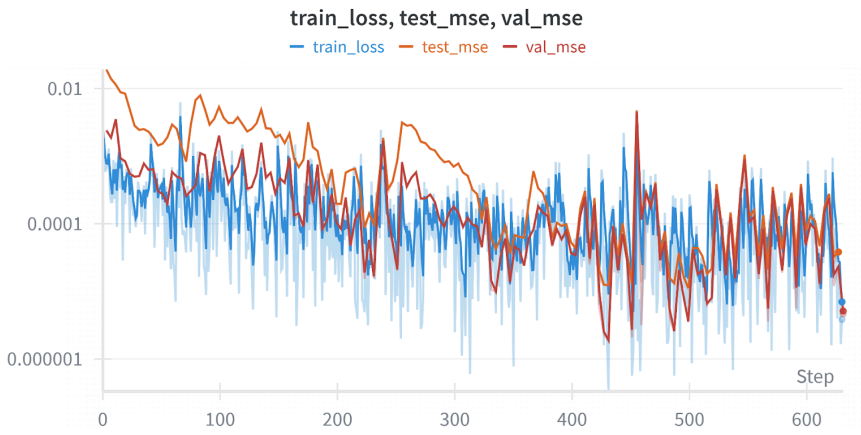


FIGURE 27:EVOLUTION OF THE LOSS FOR THE TRAINING SET, THE VALIDATION SET (THAT IS USED FOR EARLY STOPPING AND TO CHOOSE THE BEST MODEL, AND THE TEST SET)

In Figure 28, we note a slight increase of error, which is indicative of temporal distribution shift. This increase is observable (although reduced to a few samples), and during the implementation of the model, one needs to be aware of the decreasing quality for periods very different from the training set.

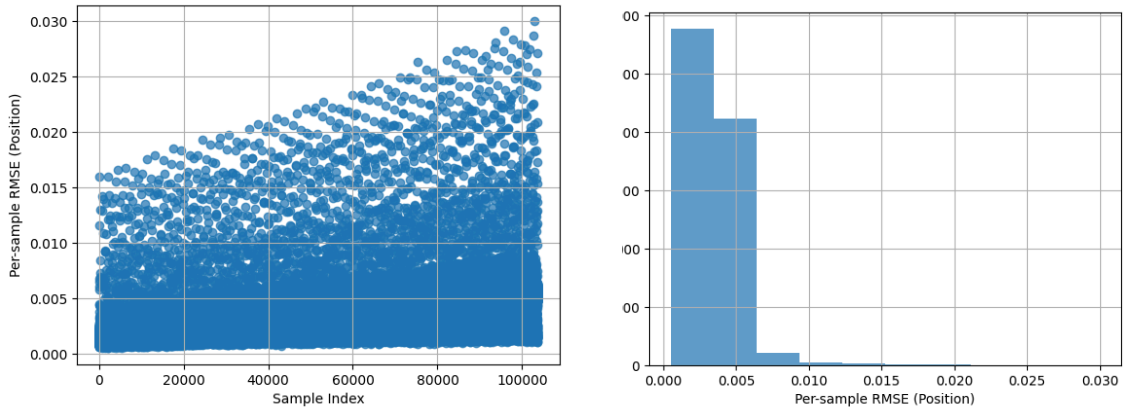


FIGURE 28: (LEFT) ERROR AS THE DISTANCE FROM THE TRAINING PERIOD INCREASES, (RIGHT) HISTOGRAM RESULT IN KM

The histogram in Figure 28 indicates that error for most predictions is below 5 metres.

## 6.3 EDP USE CASE

In the final year of the TaRDIS project, the ML development for the energy use case which was focused on the smart home environment (forecasting of energy generation and energy production) and the energy management algorithm based on Deep Reinforcement Learning

(DLR) was extended to a multi-home scenario, based on the mathematical modeling presented in D5.1 [1] and D5.2 [2].

In this context, a market-based energy exchange mechanism with dynamic price setting is presented, as well as two methods to mitigate potential market manipulation from the prosumers (smart homes that participate in the P2P energy exchange and trading framework) [52]. The configuration considered is shown in Figure 29. At each time slot t, every smart home sets a normalized selling price for its individual energy surplus, which is then scaled by the current grid price to determine the actual selling price. A threshold-based mechanism is implemented by imposing a hard upper limit to the selling energy price.

The P2P trading mechanism categorizes homes into those with energy surplus and those with energy deficit at time t. Surplus homes are ranked according to ascending selling prices, creating an ordered set. The mechanism then matches deficit homes with surplus homes, beginning with the lowest-priced surplus energy and continuing until either all deficit is covered or all surplus is exhausted. All transactions, including any grid-related fees, contribute to the overall reward structure for each smart home.

Complementing the threshold-based approach, a reward-based mechanism is additionally developed in this work, targeting further discouragement of energy price manipulation by households' intelligent agents. This method continuously monitors historical pricing data and market indicators to assess deviations from competitive market norms. When indicators such as low price variance, excessive price uniformity, or high correlation among submitted prices are observed, penalties are imposed that reduce the net rewards for the implicated smart homes. By integrating these two strategies, the proposed framework not only optimizes local energy exchange but also promotes fair pricing and prevents manipulative trading behavior.

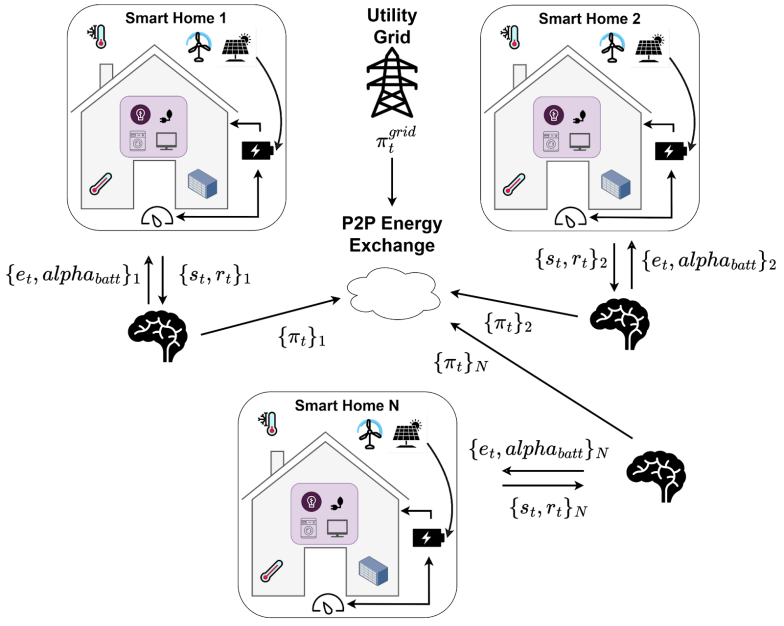


FIGURE 29: CONFIGURATION OF MULTIPLE SMART HOMES AND THE P2P ENERGY EXCHANGE FRAMEWORK

The Deep Reinforcement Learning (DRL) agent (Figure 29) of each smart home performs local actions regarding the Heating, Ventilation and Air Conditioning (HVAC) and Energy Storage System (ESS) operation, while also setting the price of excess energy in the P2P market.

During training, the proposed multi-agent DRL optimization algorithm the DRL agents interact with the environment over a sequence of discrete time steps. At each time step t, each smart home intelligent agent: (i) Observes the current state of the environment; (ii) Selects an action according to its policy; and (iii) Receives a reward and observes the next state. The DRL agent's objective is to learn a policy that maximizes the expected cumulative reward. In the smart home energy management environment of Figure 29, the interactions of each DRL agent are

- ***State***: The state of each smart home environment is represented by a vector that encapsulates both local operating conditions and global market metrics. This comprehensive representation enables the DRL agent to make informed decisions regarding energy management and price setting. The state vector, consists of seven primary features and an additional selling price vector, equips the DRL agent with the necessary information to dynamically regulate energy consumption, manage storage operations, and set competitive selling prices in a complex, multi-agent environment. According to the modeling presented in D5.2, the state includes the renewable generation output, the non shiftable power demand, the Energy Storage System (ESS) energy level, the outdoor and indoor temperatures, the dynamic grid price and the dynamic energy price set by the other smart homes.
- ***Action***: The intelligent DRL agent regulates each individual smart home environment by selecting a vector of three continuous actions at each time slot t. These actions are defined as follows:
    - HVAC Control Action: A continuous control variable in the range [0, 1] that is subsequently scaled to determine the actual HVAC power.
    - Battery Management Action: A continuous variable in the range [−1, 1] that governs battery operations; positive values correspond to charging and negative values to discharging.
    - Price-Setting Action: A normalized selling price for P2P energy trading.
- ***Reward***: The reward function is designed to balance the trade-offs between energy consumption costs, equipment degradation and trading profits, thereby guiding the DRL agent toward optimal operational strategies.

We adopt the Deep Deterministic Policy Gradient (DDPG) method for the energy management framework due to its inherent capacity to handle continuous state-action spaces, which is essential for precisely controlling HVAC systems, battery operations, and price setting tasks where discretization would lead to suboptimal performance [53]. The training process follows an episode-based loop in which the agent interacts with an environment simulating multiple houses. The overall training process for the multi-house system incorporating both the threshold-based and reward-based market control mechanisms can be described for each time step:

- Each DDPG agent observes the household-specific environment, forming an aggregated state vector.
- The intelligent agent selects actions for each house, incorporating exploration noise in the form of an Ornstein–Uhlenbeck process [54].

- The selected actions are implemented on the environment, updating the new state (energy balance, temperature, etc.), simulating P2P trading, and computing the received reward based on the selected market control mechanism.
- Transition tuples are stored in the replay buffer of each smart home and, periodically, mini-batches are sampled to update the critical networks via gradient descent.
- Then, the sampled policy gradient is used for the update of the actor networks.

To demonstrate the multi-agent environment, we simulate 18 distinct scenarios for each of the three market mechanisms, i.e., the threshold-based, reward-based, and a baseline method with no market control for N = 10 houses. These scenarios encompass variations in reward stability, trading optimization, neural network architecture, learning parameters, battery characteristics, comfort control, and multi-objective tradeoffs. The presented results consider the average values across the multiple scenarios in terms of indoor temperature control, energy management related to the ESS operation, combination of the energy consumption patterns (renewable, P2P or provided by the grid) and the learned policy of the DDPG agent for setting the P2P energy price. It is worth noting that the configuration parameters of the results presented herein are indicative and, in general, they can be directly modified on the open-source code to reflect diverse simulation scenarios, for instance different consumption profiles of the smart homes, different ESS characteristics, etc [55]. To showcase the performance of the proposed DDPG-based approach, we compare its performance against Deep Q-Network (DQN), a widely adopted discrete-action reinforcement learning algorithm

### 6.3.1 Comparative learning performance

Figure 30 shows the learning curves for both DDPG and DQN algorithms, depicting the average reward evolution over the course of the training episodes. Evidently, DDPG demonstrates faster convergence to higher reward values, while DQN converges more gradually. The performance gap of approximately 20-25% highlights DDPG's advantage in handling the continuous action space required for simultaneous HVAC control, battery management, and dynamic price setting of the multi-home configuration. The smoother convergence trajectory of DDPG also indicates more stable policy learning, which is crucial for reliable deployment in real-world smart home environments.

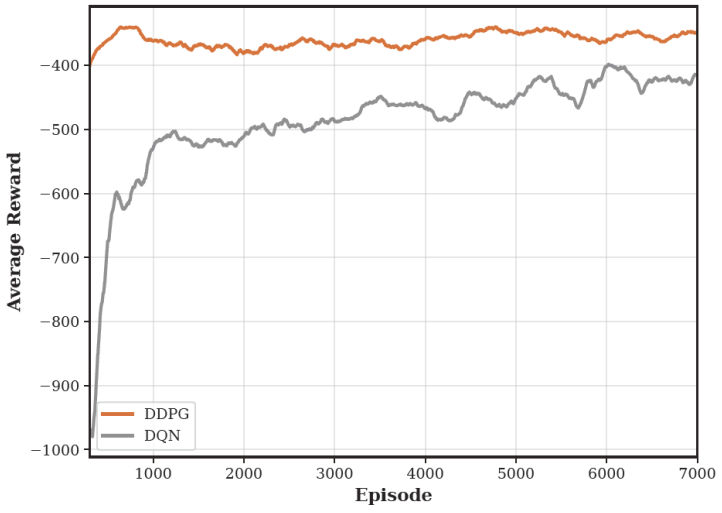


Figure 30: Learning curves for the multi-agent DDPG and DQN algorithms

Moreover, to evaluate the effectiveness of both DDPG and DQN algorithms in learning optimal pricing strategies that avoid market manipulation, the final normalized P2P prices achieved across the three market control mechanisms are shown in Figure 31 for the three market mechanisms. The reward-based mechanism achieves the lowest P2P energy prices, illustrating that the market manipulation has been prevented. In addition, DDPG consistently outperforms DQN by 15-26% across all mechanisms. It should be noted that lower prices benefit community economics and prevent market monopolization, ensuring equitable access to local renewable energy.

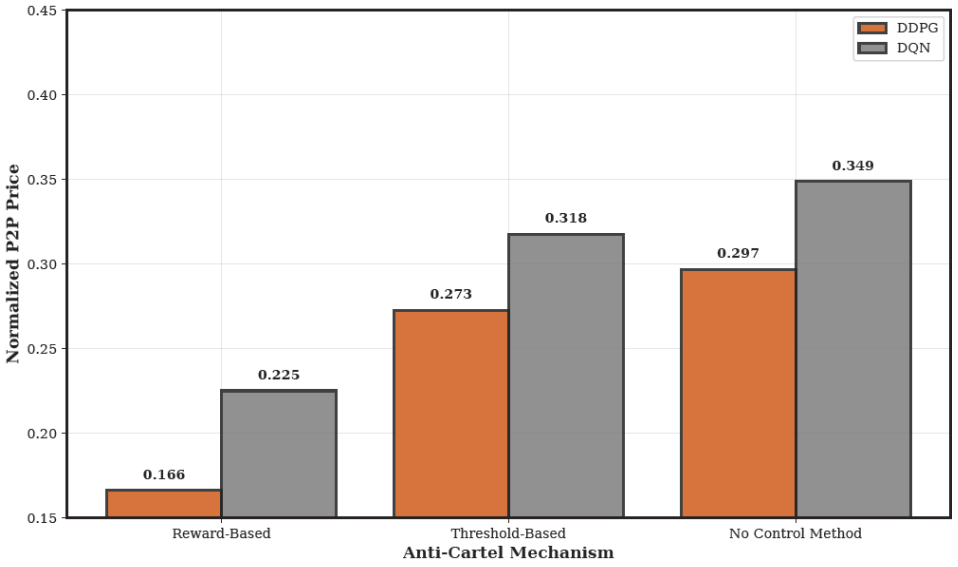


FIGURE 31: NORMALIZED P2P PRICE COMPARISON BETWEEN DDPG AND DQN ACROSS THE THREE MARKET CONTROL MECHANISMS

### 6.3.2 INDOOR TEMPERATURE CONTROL

To showcase that the intelligent DRL agents also maintain a comfortable environment inside the household, the indoor temperature profiles over a 24-hour period for both DDPG and DQN algorithms under the reward-based market control mechanism are depicted in Figure 32. The shaded green area indicates the comfort bounds (20–22°C), while the dashed gray line shows the temporal evolution of the outdoor temperature (5–20°C). Evidently, both algorithms maintain indoor temperature within the desired range for the majority of the day, while the DDPG model provides a more fine-grained temperature control as compared to the DQN model.

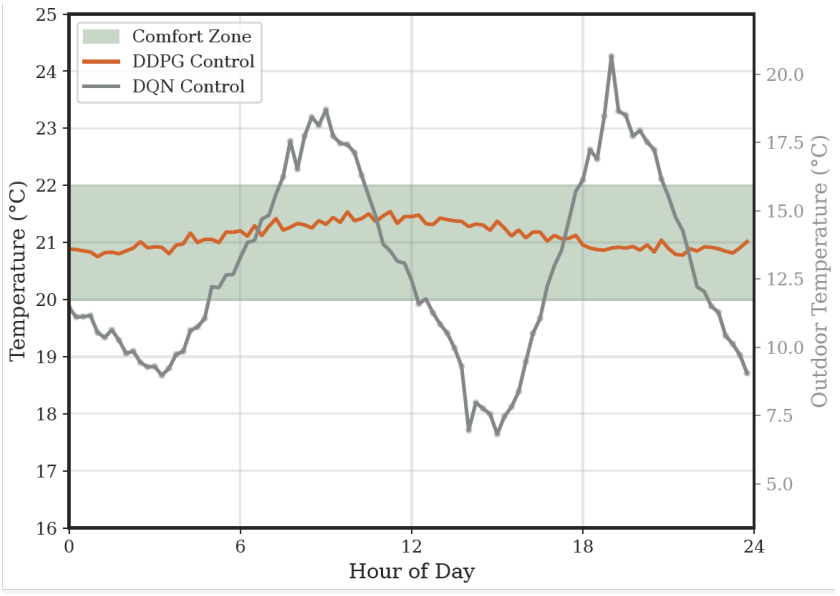


FIGURE 32: INDOOR TEMPERATURE VARIATIONS OVER 24 HOURS COMPARING DDPG AND DQN CONTROL

### 6.3.3 PERFORMANCE OF THE ENERGY STORAGE SYSTEM

Furthermore, the dynamics of the ESS in the proposed modeling framework are shown in Figure 33, in the form of charging and discharging patterns (averaged across the simulation scenarios and the N = 10 smart homes) during a 24-hour period for the DDPG and DQN algorithms. The DDPG agent demonstrates good synchronization with the grid price, performing charging during low-price periods and discharging during peak-price intervals, while the DQN model exhibits suboptimal performance, with the State of Charge (SoC) peaking at approximately 70% and maintaining higher residual charge levels (above 40%) during peak-price periods, indicating suboptimal energy utilization.

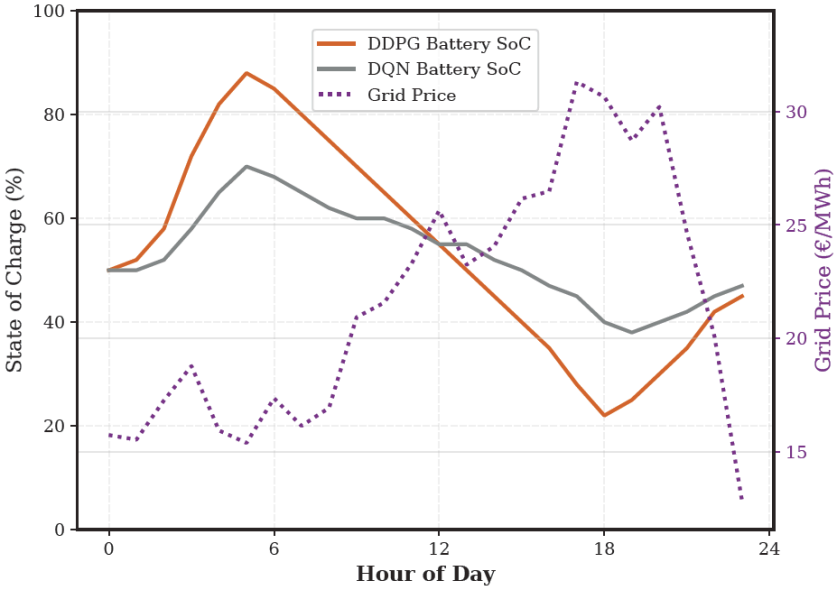


FIGURE 33: BATTERY MANAGEMENT PERFORMANCE COMPARISONS

### 6.3.4 Merged energy consumption patterns

The results in this subsection demonstrate the average energy consumption patterns for the smart homes in terms of originating source using the DDPG-based control system. In this context, each smart home uses a combination of energy gathered by renewable sources, energy provided by the grid and energy traded inside the community to cover its individual demands. The target of the multi-agent DDPG algorithm is to increase the use of the power that is produced locally (either in the smart home or traded through the energy community) and decrease the dependence on the grid power.

The average combination of the provided energy is depicted in Figure 34, integrating daily energy consumption profiles from all three control mechanisms. Specifically, the results showcase that the proposed approach manages to regulate the P2P energy market towards the general goal of sustainability, achieving a minimal footprint within the energy community by market regulation. The energy exchange within the community effectively reallocates the energy among the prosumers depending on their demands, dynamically configuring the price between the producers (that sell their surplus energy) and consumers (that buy the required energy deficit), while boosting the utilization of P2P energy that is gathered by renewable sources and minimizing the dependence on the grid.

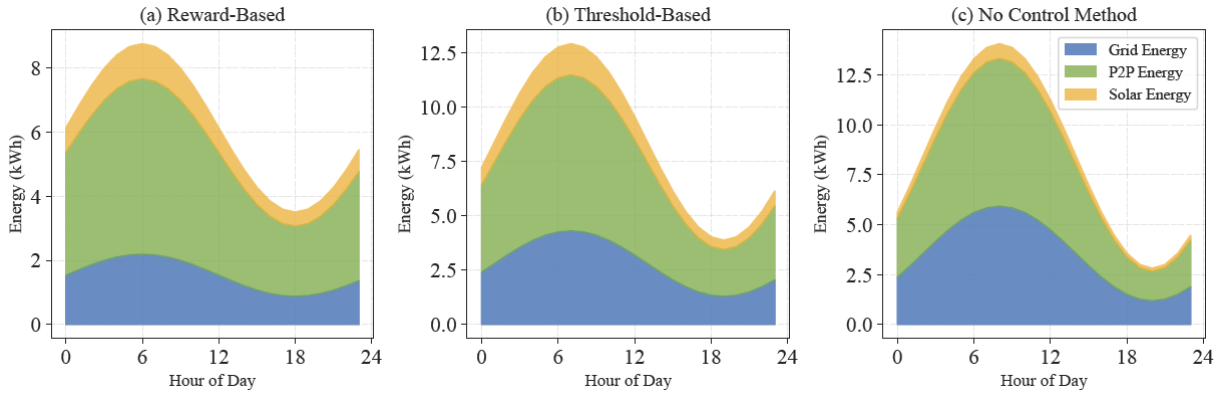


Figure 34: Merged Energy Consumption Patterns under DDPG control: Stacked area chart displaying grid, P2P, and solar energy contributions over 24 hours

## 6.4 TID use case

TID's use case focuses on the deployment of distributed learning methodologies within the context of intelligent homes, where heterogeneous devices operate as components of an automated ecosystem that enhances daily living. In this environment, characterized by a network of devices equipped with local machine learning training capabilities, TID applies its Federated Learning as a Service (FLaaS) framework (T-WP5-10) to enable the collaborative and privacy-preserving distributed training of models across multiple interconnected devices.

The implemented technology is evaluated within a task of Wake-up-word (Wuw) detection, which enables collaborative and personalized training of Wuw for speech interfaces. Wuw detection serves as an ideal benchmark for distributed learning frameworks due to its sensitivity to privacy and efficiency. Unlike centralized approaches that require uploading raw voice data to a server, FLaaS (T-WP5-10) ensures that user data remains localized while still enabling robust collaborative training. This paradigm is further improved through (i) the integration of Split Learning (SL), which lowers device-side computational demands,

(ii) Knowledge Distillation (KD), which compresses models for efficient deployment on constrained devices, and (iii) Differential Privacy (DP), which guarantees strong privacy protections against information leakage. Together, these techniques align FLaaS (T-WP5-10) with the regulatory and societal demands for privacy-preserving AI in domestic environments, while also ensuring scalability and deployment feasibility across heterogeneous devices.

Prior to the execution of TaRDIS, the FLaaS framework (T-WP5-10) was limited to supporting conventional Federated Learning (FL) settings [56]. Through the advancements introduced in TaRDIS, FLaaS (T-WP5-10) has evolved into a more comprehensive distributed learning platform, now encompassing FL, SL, and KD.

SL allows the partitioning of a neural network model between the client and the server. By delegating the computationally intensive portions of the model to the server, SL significantly lowers the resource requirements on client devices. This makes it particularly advantageous in scenarios involving heterogeneous or resource-constrained devices, such as IoT nodes or low-power edge devices, while still maintaining collaborative training capabilities across the network.

KD (T-WP5-07), in contrast, focuses on transferring the knowledge acquired by a high-capacity model (the "teacher") into a smaller, lightweight model (the "student"). This process not only reduces inference complexity and memory footprint but also enables the deployment of efficient models on devices with limited computational capabilities, without a substantial loss in accuracy. Within FLaaS (T-WP5-10), KD (T-WP5-07) plays a complementary role by facilitating model compression to foster communication efficiency.

By integrating FL, SL, and KD (T-WP5-07) within a unified platform, FLaaS (T-WP5-10) extends beyond traditional federated learning paradigms, providing a flexible and scalable solution for distributed training in diverse real-world scenarios. This evolution enables the coexistence of multiple learning strategies that optimize for factors such as privacy, communication efficiency, computational capacity, and deployment feasibility.

In previous deliverables (D2.2 [28], D5.1 [1] and D5.2 [2]), we discussed how the heterogeneity (in terms of computation, energy, memory) of these devices can be overcome through the Split Learning paradigm and Knowledge Distillation (see Section 4.3 and previous deliverables D5.1 [1] and D5.2 [2]). In this deliverable, we give updates on integration efforts for these aspects, as well as ongoing research.

### 6.4.1 FLaaS (T-WP5-10) integration with Knowledge Distillation (T-WP5-07)

The main goal of Knowledge Distillation (T-WP5-07) in the TID use-case is to improve inference performance and communication efficiency in a distributed setting. Specifically, KD allows for the training of a large, accurate teacher model at the server side, and the subsequent generation of a smaller, lightweight student model. The distilled student model can then be deployed to edge devices for inference or fine-tuned with local data.

The integration involves:

1. Server-side KD: Training a large teacher model on the server, performing KD to extract a smaller student model.
2. Model distribution: Sending the distilled student model (or parts of it) to devices for local personalization and fine-tuning.
3. Device-side deployment: Ensuring that even constrained devices can run efficient inference locally, without the need to handle full-sized models.

This approach addresses the challenges of device heterogeneity by reducing the computational footprint of models while retaining high accuracy, enabling more devices to participate in distributed learning and inference.

Figure 35 shows the integration of KD in the FLaaS framework (T-WP5-10). A large teacher model is trained at the server side, distilled into a lightweight student model, and then partially deployed to edge devices for inference and continued training. This enables **resource-constrained clients** to still contribute to collaborative intelligence without overburdening their computational capabilities.

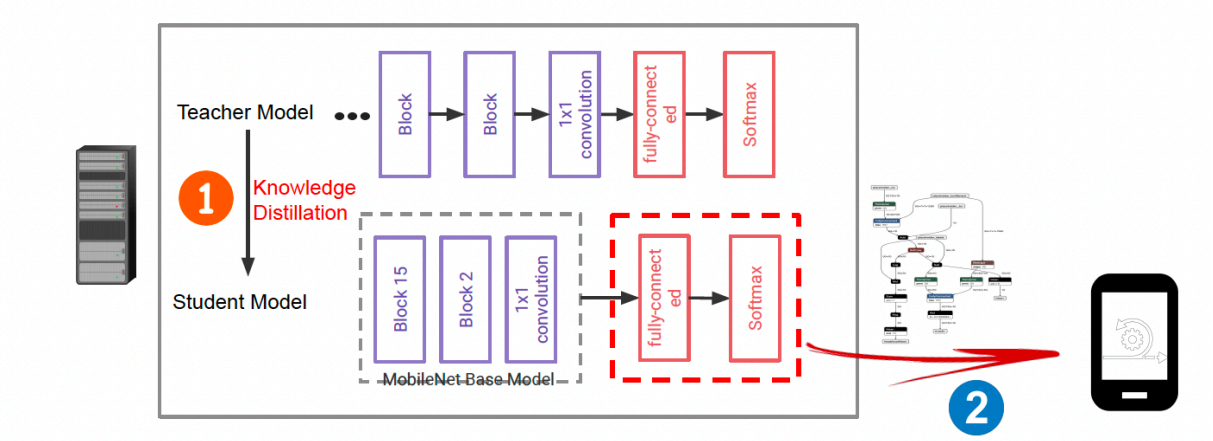


FIGURE 35: KD ARCHITECTURE DEPICTING THE LAYERS OF THE TEACHER AND STUDENT MODEL

In addition, small-scale demos have been implemented using MobileNet as the student model, with performance measurements showing reduced inference time and energy consumption on IoT devices, while maintaining acceptable accuracy. Snippets of these demos (logs and screenshots of device inference) can be included here as proof of concept.

More precisely, we implemented KD in the FLaaS (T-WP5-10) platform to reduce communication overhead. KD is a model compression technique where a large teacher model transfers its knowledge to a smaller student model, which can then be distributed to clients with much lower transmission cost while maintaining similar accuracy. Within the FLaaS admin interface, the use of KD can be enabled directly as a project option, allowing practitioners to reduce transmission overhead without modifying the training logic.

In our evaluation, the FedAvg baseline required sending the full global model of 78.53 Megabytes (MB) (82,335,703 bytes) to each client per round, whereas KD reduced this to a distilled student model of 8.76 MB (9,186,443 bytes), corresponding to an 88.85% reduction in transmission overhead.

For power verification, we considered the server-to-client downlink, which dominates the transmission cost, and used the server-to-S3 upload as a proxy. The estimated energy consumption is shown in Figure 36.

Funded by the European Union

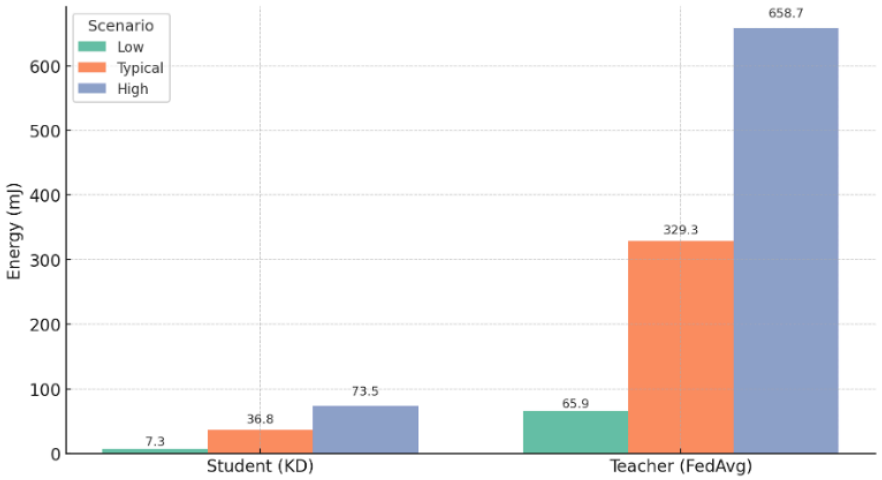


FIGURE 36: ENERGY CONSUMPTION FOR TRANSMITTING MODEL WEIGHTS, COMPARING THE DISTILLED STUDENT MODEL (KD) AGAINST THE FULL TEACHER MODEL (FEDAVG)

KD preserved accuracy while significantly reducing transmission cost. Overall, deploying knowledge distillation in the FLaaS server lowers per-round transmission overhead by ~88.84%, consistently across both byte-level and energy-based estimates.

We implemented KD in the FLaaS platform to also evaluate model compression. KD is a technique where a large teacher model transfers its knowledge to a smaller student model, which can then be distributed to clients with substantially reduced size and resource footprint while maintaining comparable accuracy.

The teacher model was exported with a size of 78.53 MB (82,344,211 bytes), while the distilled student model had a size of 8.76 MB (9,186,443 bytes), corresponding to a compression rate of 88.8%. In addition to the reduction in model size, we profiled the server-side transmission process to quantify resource savings. Results are shown in Table 21 below.

TABLE 21: PERFORMANCE RESULTS OF KD IN FLAAS

| Model | Size (MB) | Wall Time (s) | CPU Time (s) | Avg CPU % | Peak RSS (MB) | Energy Typ (mJ) |
|---|---|---|---|---|---|---|
| Teacher | 78.53 | 4.354 | 4.443 | 102.5 | 508.22 | 329.38 |
| Student | 8.76 | 1.522 | 0.210 | 18.6 | 345.31 | 36.75 |

### 6.4.2 FLaaS (T-WP5-10) extension to support Split Learning

The incorporation of SL in TID's use case offers a key feature designed to offload part of the training workload from resource-constrained clients to a server. This approach is particularly useful when client devices lack sufficient computational resources for full on-device training.

As introduced in the previous deliverable D5.2, in SL, a Deep Neural Network (DNN) model is divided into two or more sequential segments: part-1 and part-2. The last layer of part-1 is referred to as the cut layer. During training, part-1 is processed on the client, while part-2 is processed on the server, enabling collaborative training without exposing raw client data.

Among the various SL implementations, the implemented architecture focuses on Split Federated Learning (SFL), also known as SplitFedv2 [57]. SFL integrates the distributed aggregation strategy of Federated Learning (FL) with the architectural partitioning of SL. In SFL, clients train part-1 locally and aggregate their models at the end of each communication round, while the server trains part-2 by sequentially processing intermediate activations received from clients. This is represented in Figure 37. Representation of SL split with two model parts.

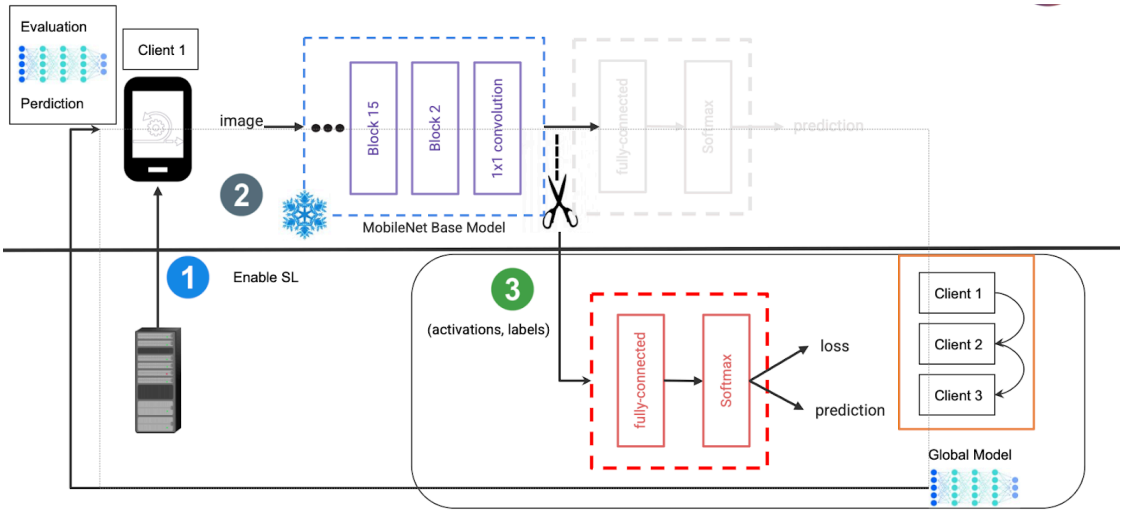


FIGURE 37: REPRESENTATION OF SL SPLIT WITH TWO MODEL PARTS

FLaaS enables two distinct SL execution modes: i) without SL helpers; ii) with SL helpers;. The execution without helpers depends exclusively on the server, which sequentially trains model part-2 for each client and performs the aggregation. On the other hand, helpers in SL are computational agents that handle the server-side training portion of the split neural network, but they differ fundamentally from the traditional centralized server component. Unlike the classic server, which acts as the main trainer for part-2, coordinator and aggregate for model updates, helpers focus on the training of model's part-2 and can be distributed across multiple entities and serve as decentralized compute resources within a collaborative network. Helpers can be used to alleviate the computing workload on the server by distributing the training of model's part-2. In FLaaS, helpers were implemented using Docker and FastAPI for easy instantiation and deployment on a remote computing infrastructure, which fosters scalability.

Figure 38 represents the execution workflow of FLaaS (T-WP5-10) without SL helpers and with SL helpers respectively.

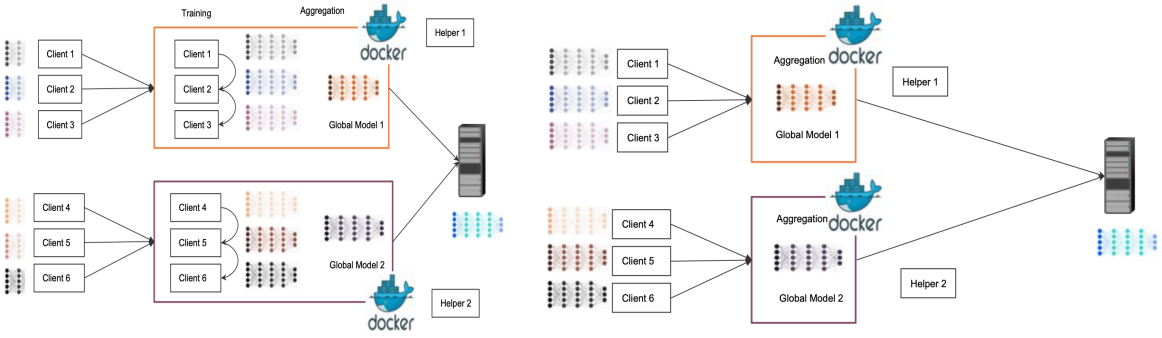


FIGURE 38: FLAAS EXECUTION WORKFLOW OF SL WITHOUT HELPERS (LEFT) AND USING HELPERS (RIGHT)

To validate the implemented architecture of SL in FLaaS, we evaluate the gain in terms of Random Access Memory (RAM) usage on the client side during training.

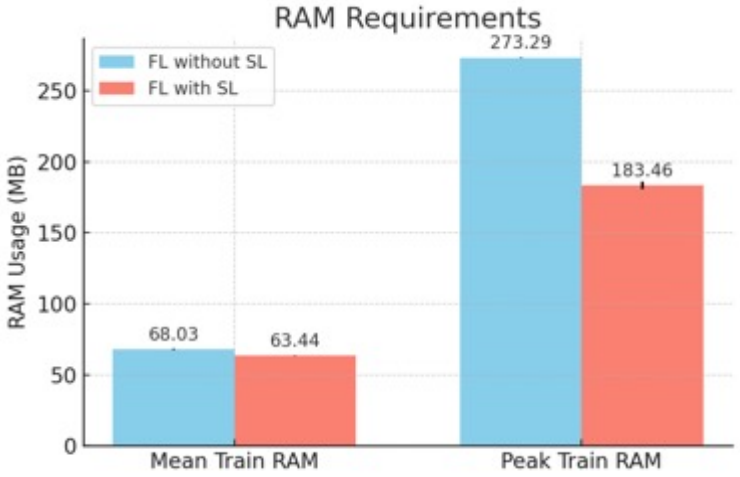


FIGURE 39: COMPARISON OF MEAN AND PEAK CLIENT RAM USAGE DURING FL TRAINING WITH AND WITHOUT SPLITTING (SL)

The comparison results, shown in Figure 39 (RAM), demonstrate that SL provides substantial reduction of RAM usage, particularly peak consumption during training. This makes SL well-suited for deployment on resource-constrained devices, such as smartphones or IoT hardware, where memory limitations often hinder the execution of large models in FL.

### 6.4.3 FLaaS (T-WP5-10) extension to support Differential Privacy

The main goal of DP in the TID use case is to provide formal privacy guarantees for client contributions during distributed training, while keeping model utility within acceptable bounds. Concretely, DP limits the influence of any single user's data on the learned model by (i) clipping per-update sensitivity and (ii) injecting calibrated noise during aggregation or before transmission.

More precisely, the integration involves:

- **Central DP:** Clients send raw updates and the server applies $\ell_2$-clipping on the aggregated update and adds Gaussian noise before updating the global model.
- **Local DP:** Each client clips its own update and adds Gaussian noise locally prior to transmission, removing the need to trust the server at the cost of higher noise.

The DP mechanism is parametrized by the following parameters. We configure privacy budget (ε,δ), clipping norm C, and noise multiplier σ. FLaaS (T-WP5-10) derives σ automatically from (ε,δ), sampling rate, and number of rounds, or accepts explicit values.

The figure below illustrates both Central and Local DP workflows in FLaaS (T-WP5-10), including the points at which clipping and noise are applied, and where accounting occurs. This is represented in Figure 40.

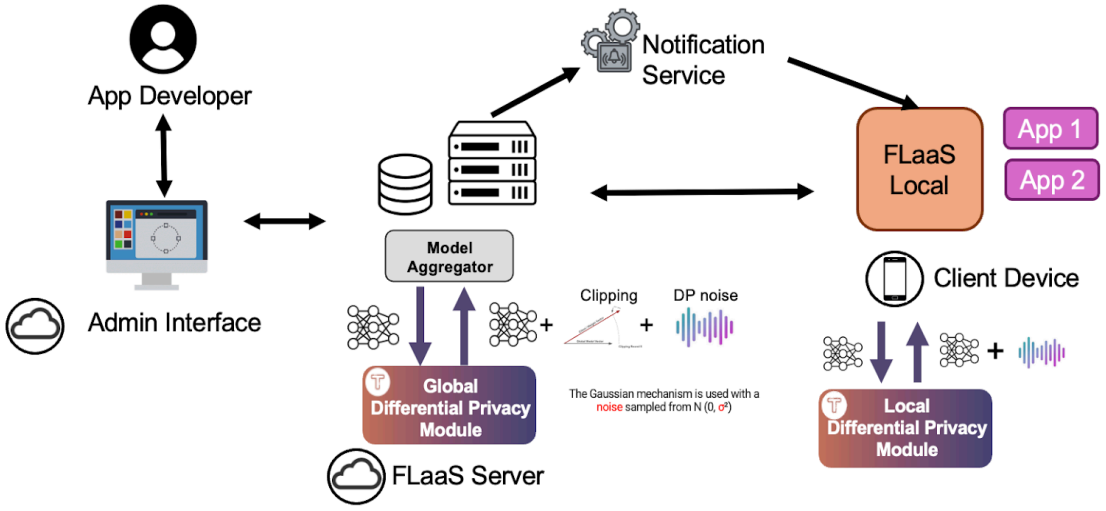


FIGURE 40: FLAAS EXECUTION WORKFLOW OF DP

In central DP, clients send unclipped model updates to a trusted server that aggregates them, clips the aggregate to a fixed $\ell_2$ norm, injects calibrated Gaussian noise, and then applies the noisy aggregate to the global model. Because the noise is added once after aggregation, central DP generally provides stronger utility for a given privacy budget ε. In contrast, local DP requires every client to clip its own update and add noise before transmission, removing the need to trust the server but causing noise to accumulate with the number of clients. Consequently, at the same ε, local DP typically yields lower accuracy.

To quantify this trade-off, we conducted a series of experiments where FL was carried out under both DP mechanisms. The privacy budget ε was varied over the values {0.2, 0.4, 0.6, 0.8}, with a fixed $\delta = 10^{-5}$. A No DP baseline (no noise injection) was included for reference. Each configuration was run for 10 communication rounds and repeated three times with different random seeds to ensure robustness. We recorded the mean accuracy and the full range (min–max) across the three runs. The results are summarized in Figure 41.

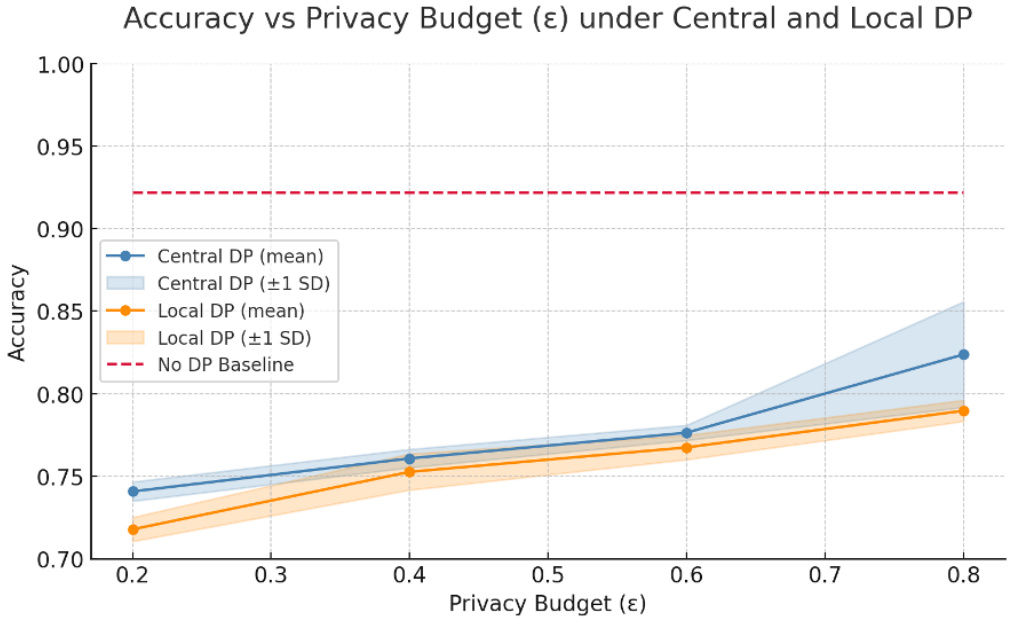


FIGURE 41: ACCURACY VS. PRIVACY BUDGET (ε) UNDER CENTRAL AND LOCAL DP

Our results confirm the expected monotonic relationship between ε and model utility. Central DP consistently outperformed Local DP at all privacy levels due to less noise accumulation. At ε = 0.2, Central DP retained approximately 85% of the No DP accuracy, while Local DP achieved only 78%. As ε increased, accuracy improved for both methods. Notably, at ε = 0.8, Central DP reached ~95.5% of the No DP baseline, compared to ~85.7% for Local DP.

These findings support the objective of this KPI, which aims to study how the integration of privacy-enhancing mechanisms affects model performance in federated learning. Central DP, while requiring a trusted server, provides a more favorable privacy-utility trade-off. Local DP enhances decentralization and trust minimisation but demands higher ε values to achieve comparable utility. These insights inform the design of privacy-aware FL systems based on trust assumptions and application requirements.

### 6.4.4 Fundamental research on planning and lightweight Machine Learning

In this subsection, we discuss fundamental algorithmic research in planning and lightweight machine learning that we have performed with the aim of future improvements in the efficiency of FLaaS (T-WP5-10). This research has culminated in two publications, one in International Joint Conference on Artificial Intelligence (IJCAI) 2025 [58] and one in NeurIPS 2025 [59]. Specifically, the first covers hierarchical task network planning, and the second investigates lightweight machine learning via the VC-dimension. Here, we provide a high-level overview of the highlights of each of these papers. Further definitions, results, and details can be found in the papers themselves.

Due to the broad connectivity in next-generation networks like 6G, the importance of planning is ever-increasing. This is particularly the case for the orchestration of tasks that need to be accomplished by many devices. Such tasks often have precedence constraints, causes, and effects, necessitating a proper planning strategy for successful execution. Hierarchical task networks (HTNs) were introduced as a planning framework that abstracts away the nature of the tasks to be executed and how they are to be executed, allowing for the same planning analysis to apply to many different scenarios.

HTN planning problems generally consist of a state space (describing the current state of the system), a partial order over a set of tasks (determining precedence constraints for the tasks) that can be represented through a directed graph, and an action associated to each task that has state space prerequisites for it to be executed, as well as effects on the state space if it is executed. Examples of HTN planning problems are plan verification, plan existence, action executability, and state reachability. It is well-established that HTN planning problems are NP-hard [60], and in fact, the latter three are even NP-hard when the partially ordered set is an antichain (i.e., any two distinct elements are incomparable), due to the large state space. To overcome this intractability, we exploit structural properties of the input digraph to design polynomial-time algorithms for all four of the above problems (for the latter three problems, the state space must also be bounded). Specifically, when (1) the generalized partial order width (i.e., the maximum number of pairwise-disjoint maximal chains in a partially ordered set when ignoring isolated elements) or (2) the vertex cover number (i.e., the minimum number of vertices to cover all the edges of the digraph) are fixed (i.e., a constant that does not depend on the input size), then we obtain polynomial-time algorithms for all four problems.

As mentioned above, the second work focuses on exploiting the VC-dimension for lightweight machine learning. The VC-dimension is a complexity measure of the expressivity of a set system (i.e., binary concept class) that provides theoretical guarantees on the complexity of machine learning models, their performance, how much data is required to train them, and the error rate on new data. In a nutshell, machine learning is

more efficient when the VC-dimension is small. Thus, in the pursuit of lightweight machine learning for binary classification, computing the VC-dimension can play a vital role.

Given a non-empty finite set $V$ and a set system $C \subseteq 2^V$, the VC-dimension of $C$ is the size of a largest set $S \subseteq V$ that is shattered by $C$, i.e., such that $\{X \cap S : X \in C\} = 2^S$. Naturally, the VC-dimension problem is often formulated via a hypergraph $H = (V, E)$, where the vertex set $V$ is the domain as above, and the hyperedge set $E$ is the set system $C$ as above. It is well-known that the VC-dimension can be computed in quasi-polynomial time, but the problem is LogNP-complete [61], and thus, it is unlikely for it to be efficiently computable (i.e., in polynomial time). To circumvent this intractability, we provide algorithms that compute the VC-dimension in polynomial time when certain parameters of the input are fixed (i.e., a constant that is independent of the input size). Specifically, we design two algorithms: an exact algorithm computing the VC-dimension in $2^{O(tw \cdot log(tw))} \cdot |V|$ time and a 1-additive approximation algorithm that computes either the VC-dimension or VC-dimension plus one in $2^{O(\Delta \cdot log(\Delta))} \cdot |V + E|^{O(1)}$ time, where $\Delta$ is the maximum degree of $H$ and $tw$ is the treewidth of the bipartite incidence graph $G$ that is an equivalent representation of the hypergraph $H$. Importantly, when $\Delta$ or $tw$ is a constant, these algorithms run in polynomial time.

# 7 Contribution to TaRDIS KPIs

This section contains a report on the WP5 contributions to KPIs, and an overview of milestones and objectives related to the efforts of WP5. We first briefly discuss the milestones and objectives, focusing on the ways WP5 met them. Then, we show how the different tools addressed the targeted KPIs. We focus on the requirements (see Section 5.1 above) and corresponding KPIs that are relevant for WP5. The AI/ML tools also address various use case specific requirements and KPIs, but these results will be reported in upcoming WP7 deliverables, due to their direct linkage with the use cases.

## 7.1 Project milestones and objectives

In this section, we provide an overview of how WP5 meets the project objectives and milestones.

### 7.1.1 Project milestones

The TaRDIS project identifies four milestones (MS), according to the GA:

- MS1: Requirements
- MS2: Proof of concept
- MS3: Prototype
- MS4: TaRDIS

WP5 contributes to MS2, MS3 and MS4. A proof of concept represents a demonstration that an idea or approach works in practice, meaning that a solution is feasible, even if not fully developed. D5.1 [1] is a means of verification for this milestone. It contains an initial overview of the approaches in development for each task, supported by research and some early tests. We can already identify the outlines and bases for future WP5 TaRDIS tools here. Also, the initial ML modeling ideas for the use cases are presented in D5.1 [1].

A prototype is meant to show that tools and technologies are working, with operational core functionalities. D5.2 [2] is a means for verification for this milestone. It represents an overview of working tools that are part of the TaRDIS architecture. The tools are demonstrated by corresponding evaluations, where some of them already address a set of relevant KPIs. The ML models for the use cases are elaborated and demonstrated in detail.

The final milestone, TaRDIS, represents a point where the developments and evaluations are finished and consolidated into a distributed programming toolbox. In the context of WP5, this means that all the tools are developed and ready to be used. Since WP5 finishes earlier than the project lifespan, it is anticipated that the WP5 tools will be utilized further, in order to meet the project's expected outcomes. This deliverable represents a means of verification for the final milestone, where all WP5 tools are developed and placed properly in different aspects of TaRDIS.

### 7.1.2 Project objectives

In this section, we summarize the contributions made regarding the overall TaRDIS and WP5 specific objectives per GA. In D5.2 [2], we already discussed the relevant objectives and results that are met or planned to be met. To recall, the project objectives of interest for WP5 are the following:

- *Objective 1*: Novel programming model for heterogeneous swarms

- *Objective 3*: Decentralized intelligence for heterogeneous swarms
- *Objective 5:* Interoperable execution environment

In Table 22, we list the connection of AI/ML with the TaRDIS objectives and results.

TABLE 22: WP5 CONTRIBUTIONS TO TARDIS OBJECTIVES AND RESULTS

| TaRDIS objective | Result | AI/ML tools |
|---|---|---|
| Objective 1 | R1.2 APIs for distribution, data management, and AI/ML | PTB-FLA and MPT-FLA (T-WP5-04) |
| Objective 3 | R3.1 Techniques, algorithms, and models to support swarm intelligence. | Flower-based FL tool (T-WP5-01/02/03), FAuNO (T-WP5-05) with PeersimGym, Fedra (T-WP5-09) and Lightweight ML tools (T-WP5-06/07/08) |
| Objective 3 | R3.2 Open-source implementation of decentralized algorithms for FL and Supervised Learning (SL) | Flower-based FL tool (T-WP5-01/02/03), Fedra (T-WP5-09) and Lightweight ML tools (T-WP5-06/07/08) |
| Objective 3 | R3.3 Contextual Machine Learning Operations (MLOps) solutions for swarm intelligence. | Flower-based FL tool (T-WP5-01/02/03) |
| Objective 3 | R3.4 Dynamic peer-to-peer resource orchestration for the computing continuum. | FAuNO (T-WP5-05) with PeersimGym |
| Objective 5 | R5.1 Open and extensible development environment supporting the TaRDIS' methodology and toolbox | PTB-FLA & MPT-FLA (T-WP5-04) |

R1.2 for Objective 1 was already achieved in D5.2 [2], as the PTB-FLA and MPT-FLA tools (T-WP5-04) were already developed (See Section 7.1.2 in D5.2). However, it was planned to adapt and extend PTB-FLA to enable its usage in the GMV use case. This was accomplished and reported in this deliverable (See Section 2.2.2).

R3.1 for Objective 3 is achieved through various AI/ML tools that evolved since the submission of D5.2 [2]. The Flower-based FL tool (T-WP5-01/02/03) was enhanced and enriched with additional FL approaches (see Section 2.1). The FAuNO (T-WP5-05) orchestrator was planned to be adapted for decentralized settings, which was accomplished (see Section 3.2). Fedra (T-WP5-09) was utilized for the needs of the EDP use case, which was also planned in D5.2 [2]. The lightweight ML tools (T-WP5-06/07/08) have also evolved in the recent reporting period (see Section 4), and the intended testing of these techniques with real data is directly connected their usage in TaRDIS use cases, i.e. Knowledge distillation (T-WP5-07) in the TID use case and Pruning (T-WP5-08) in the EDP use case.

R3.2 for Objective 3 is also achieved through multiple AI/ML tools. It means providing open-source implementations of decentralized solutions for FL and SL. The Flower-based FL tool (T-WP5-01/02/03), Fedra (T-WP5-09) and the Lightweight ML tools (T-WP5-06/07/08) are meant for enabling decentralized solutions by their nature. These

tools have been upgraded during the last reporting period, so that they contain all the planned features. Besides that, the implementations of these tools are made publicly available and can be found in the appropriate repositories (the Flower-based FL tool (T-WP5-01/02/03) in [6], Fedra (T-WP5-09) in [18], Pruning (T-WP5-08) in [24], EE (T-WP5-06) in [25], KD (T-WP5-07) in [26]).

R3.3 for Objective 3 is achieved by the Flower-based FL tool (T-WP5-01/02/03), as the tool manages the full ML cycle, including preprocessing, training and inference, while guiding the user with context-aware setup options, and the FL approach directly supports the concept of swarm intelligence. As the mentioned aspects of the tool are all implemented now, this objective is met.

R3.4 for Objective 3 is achieved through the RL-based orchestration tool, FAuNO (T-WP5-05) and the decentralized ML for orbit determination in satellite swarms. The decentralized versions have been developed since D5.2 [2] was submitted.

R5.1 for Objective 5 is achieved by the PTB-FLA and MPT-FLA tools (T-WP5-04). These tools represent an open and extensible approach. They have been extended in the recent reporting period, and integrated into the GMV use case, supporting the TaRDIS methodology.

The WP5 specific objectives are defined as follows:

- *WP5 Objective 1:* Develop a framework supporting decentralized learning and inference through AI/ML programming primitives
- *WP5 Objective 2:* Exploit and specialise the preceding for the planning, deployment, and orchestration of the complete TaRDIS framework through RL and other relevant methodologies
- *WP5 Objective 3:* develop novel lightweight ML techniques to enable decentralized and swarm learning in resource-constrained devices

In Table 23, we list the connection of AI/ML with the specific WP5 objectives.

Objective 1 has been achieved in several ways. The Flower-based FL tool (T-WP5-01/02/03) was developed, in order to provide a convenient approach for supporting the complete FL process. It includes a set of possibilities to train an ML model, but also enables reprocessing and inference, while leading the user through the process of setting up the training. Also, PTB-FLA and MPT-FLA (T-WP5-04) are FL frameworks, developed to support decentralized learning and inference through AI/ML primitives. Fedra (T-WP5-09) was also designed and developed, as an additional framework that enables model-agnostic FL learning. Finally, precise ML Algorithms for Orbit Determination were developed for the GMV use case.

Objective 2 has been achieved through the development of the Federated AI Network Orchestrator (FAuNO, i.e., (T-WP5-05)), that allows RL agents under a Markov Game framework to solve the task offloading problem. Also, a training environment for decentralized agents was developed, the PeersymGym.

Objective 3 has been met in three different ways. First, the MPT-FLA (T-WP5-04) framework was developed and experimentally validated in resource-constrained environments. Next, communication efficient vertical FL via compressed error feedback was developed. Finally, three lightweight ML tools have been developed and evaluated: early-exit (EE, T-WP5-06) of inference, knowledge distillation (KD, T-WP5-07) and pruning (T-WP5-08) techniques.

TABLE 23: WP5 CONTRIBUTIONS TO WP5 OBJECTIVES

| TaRDIS objective | Contribution | AI/ML tools |
|---|---|---|
| Objective 1 | An AI/ML library of implemented FL solutions for distributed AI applications | Flower-based FL tool (T-WP5-01/02/03) |
| Objective 1 | Development environments for implementing the FL programming primitives | PTB-FLA & MPT-FLA (T-WP5-04) |
| Objective 1 | Precise ML Algorithms for Orbit Determination | GMV ML model |
| Objective 1 | A decentralized FL framework | Fedra (T-WP5-09) |
| Objective 2 | The Federated AI Network Orchestrator | FAuNO (T-WP5-05) |
| Objective 3 | A MycroPython implementation for PTB-FLA | PTB-FLA & MPT-FLA (T-WP5-04) |
| Objective 3 | Vertical federated learning, focusing on communication efficiency through compressed error feedback | Communication efficient vertical FL via compressed error feedback |
| Objective 3 | Three Lightweight ML tools methods were introduced | Early-exit (T-WP5-06) of inference, Knowledge Distillation (KD, T-WP5-07), Pruning (T-WP5-07) |

## 7.2 LINK WITH KPIS

We summarize the KPIs that are related and relevant to decentralized AI/ML tools here. The requirements were established in D2.2 [28], where a complete list of KPI was also presented. In D2.3 [3], an initial view on requirements to be fulfilled by different tools was defined. These relations matured, so that some of the requirements were changed to be connected to different, more appropriate KPIs. The decision to make these changes was made at TaRDIS General Assembly (GA) in Zurich (June 23-25, 2025). In Section 5.1 (See Table 12), we already showed the mapping between AI/ML tools and WP5 requirements. We also mentioned the KPIs that are relevant to these requirements.

In D5.2 [2], we listed the KPIs that were relevant to the AI/ML tools at that point. Some of those were addressed, and some of them are left to be addressed here. Considering the changes agreed at GA in Zurich, some parts of Table 7 in D5.2 became obsolete. In order to provide a clear and complete overview, we now list the final relevant KPIs connected to the AI/ML tools and requirements in Table 24. Note that O means Objective, and B means Baseline, in KPIs IDs, as denoted in D2.2 [28].

TABLE 24: KPIS RELEVANT FOR DECENTRALIZED ML SPECIFIC TOOLS

| ID | Description | Requirement | WP5 tools |
|---|---|---|---|
| K-O-3.1 | Use TaRDIS ML to autonomously manage system operations | RF-WP5-RLALG-01/02/03 | FAuNO (T-WP5-05) |
| K-O-3.2 | Improved edge orchestration | RF-WP5-RLALG-01/02/03 | FAuNO (T-WP5-05) |
| K-O-3.3 | Reduced Transmission overhead by 20% | RF-WP5-FLALG-04 | Flower-based FL tool (T-WP5-01/02/03)<br>Fedra (T-WP5-09)<br>Lightweight ML tools (T-WP5-06/07/08)<br>PTB-FLA (T-WP5-05)<br>FLaaS (T-WP5-10) |
| K-O-3.4 | Model reduction/compression increased by 15% | RF-WP5-FL-ALG-06 | Fedra (T-WP5-09)<br>FLaaS (T-WP5-20)<br>Lightweight ML tools (T-WP5-06/07/08) |
| K-O-3.5 | Reduced model training time by 25% | RF-WP5-FL-ALG-06 | Fedra (T-WP5-09)<br>FLaaS (T-WP5-10) |
| K-B-06 | | | Fedra (T-WP5-09)<br>FLaaS (T-WP5-20)<br>Lightweight ML tools (T-WP5-06/07/08) |
| K-B-07 | FL training latency | RF-WP5-FL-ALG-06 | *Flower-based FL tool (T-WP5-01/02/03)<br>Fedra (T-WP5-09)<br>Lightweight ML tools (T-WP5-06/07/08)<br>FLaaS (T-WP5-10) |
| K-B-08 | FL storage/RAM requirements per node | RF-WP5-FL-ALG-06 | *Flower-based FL tool (T-WP5-01/02/03)<br>Fedra (T-WP5-09)<br>Lightweight ML tools (T-WP5-06/07/08)<br>FLaaS (T-WP5-10) |
| K-B-10 | FL accuracy | RF-WP5-FL-ALG-06 | *Flower-based FL tool (T-WP5-01/02/03)<br>Fedra (T-WP5-09)<br>Lightweight ML tools (T-WP5-06/07/08)<br>FLaaS (T-WP5-10) |
| K-O-2.5 | The training time of an FL algorithm | N/A | PTB-FLA (T-WP5-04)<br>Flower-based FL tool (T-WP5-01/02/03) |

When compared to Table 7 from D5.2 [2], it can be observed that the KPI K-O-1.3 Decreased median development time by 25%, has been removed here. The reason for this is that a new collective interpretation was established for this during the GA in Zurich (June 2025). Also, according to the same renewed interpretation, K-O-3-3 Reduced Transmission overhead by 20% is no longer connected to the requirement RF-WP5-FL-ALG-01. This requirement now involves the following: A list of at least 3 FL algorithms, which has been already discussed in Section 5.1. Note also that we marked the Flower-based FL tool (T-WP5-01/02/03) with an asterisk for K-B-07/08/10. The reason is that the Flower-based FL tool (T-WP5-01/02/03) was not foreseen to satisfy the requirement that correspond to this KPI, i.e. RF-WP5-FL-ALG-06. However, the process of validating the tool naturally yields results that address these KPIs. We explain the ways of addressing these KPIs in the following subsections, for each AI/ML tool separately.

The use case-specific KPIs, which correspond to use case specific requirements, will be addressed in the upcoming deliverables of WP7.

### 7.2.1 KPIs for the Flower-based FL tool (T-WP5-01/02/03)

According to Table 24, the Flower-based FL tool (T-WP5-01/02/03) is connected to the following KPIs:

- K-O-3.3 Reduced Transmission overhead by 20%
- K-B-07 FL training latency
- K-B-08 FL storage/RAM requirements per node
- K-B-10 FL accuracy
- K-O-2.5 The training time of a FL algorithm

We now discuss each of these separately.

#### 7.2.1.1 K-O-3.3 Reduced Transmission overhead by 20%

We illustrate this KPI on the Flower-based FL tool (T-WP5-01/02/03), by taking the FedAvg algorithm as a baseline. We perform a classification task on the MNIST [62] dataset, using a CNN model, implemented in the Flower-based FL too (T-WP5-01/02/03)l. We use 2 clients for simplicity. First, we run the baseline, by using FedAvg [9]. It turns out that this setup involves 0.716 MB total per round. We compute this by observing the total number of bytes transmitted by each client, counting both directions of communication. We then summarize it for all clients (in this case 2 of them), and calculate the average per number of rounds.

Next, we apply pFedMe [63] with quantization for the same setup. The model parameters are quantized to 16-bit floats (FP16) before transmission. The reduction in the number of transmitted bytes is significant, as the total transmission per round is 0.36 MB. The transmission overhead was reduced by 49.6%. An important aspect is also the preservation of the accuracy, while applying the quantization approach. FP16 quantization was applied only to transmitted weights, while all local computations were performed in FP32 precision. This preserves numerical stability during the computations. Mixed-precision training is known to maintain comparable accuracy to full-precision models when used as described.The expected accuracy degradation is negligible.

In Table 25, we show the measures obtained for the quantized pFedMe algorithm application. We can see that each client exchanges 0.18 MB per round. The total network overhead for 5-round training is 1.8 MB, which is negligible.

Table 25: Transmission measures for 2 clients

| Measure | Value (MB) |
|---|---|
| Average client communication per round | 0.18 MB |
| Total network communication per round (all clients) | 0.361 MB |
| Total communication (5 rounds, 2 clients) | 1.805 MB |

#### 7.2.1.2 K-B-07 FL training latency

The test setup, used to demonstrate K-O-3.3, can be used to illustrate some aspects of training latency. In Table 26, we show the timings across rounds, measuring both client-side training latency and total communication overhead. The second column indicates the timing of the slowest client to perform fitting. The third column represents the server aggregation time and the fourth column is the total latency per round. The table shows a fast communication/training cycle, with average round latency 0.12s, where the dominant factor is the client-side training.

Table 26: Client and server side latencies

| Round | Max client fit (s) | Server aggregation (s) | Total latency per round (s) |
|---|---|---|---|
| 1 | 0.1601 | 0.004350 | 0.1645 |
| 2 | 0.0785 | 0.004189 | 0.0827 |
| 3 | 0.1063 | 0.005340 | 0.1116 |
| 4 | 0.1417 | 0.003712 | 0.1454 |
| 5 | 0.0900 | 0.004189 | 0.0942 |

In Table 27, we show the minimal, maximal and average FL training latencies, as well as the standard deviation. The relatively low variance of 0.03 implies stable client computation times across rounds. These timings are relative to the size of the input data and number of clients used for the FL process. The presented results illustrate the approaches we use to address this KPI. The metrics are applicable to any training setup that the Flower-based FL tool (T-WP5-01/02/03) offers.

Table 27: Latency indicators

| Latency type | Value |
|---|---|
| Average FL training latency | 0.1197 |
| Minimal latency | 0.0827 |
| Maximal latency | 0.1645 |
| Standard deviation | 0.0315 |

We also addressed some aspects of this KPI in D5.2 [2], where we showed the scaling properties of a classification task for the FashionMNIST [64] dataset, using pFedMe. The

test showed good scaling properties. We also measured timings for the Autoencoder-based anomaly detection approach on the MetroPT-3 dataset. However, the implementation was serial at that point. Now, we measure the timings for the Autoencoder-based anomaly detection on the MetroPT-3 dataset, but in a federated setup. The tests are performed in a cluster environment. The results are shown in Figure 42.

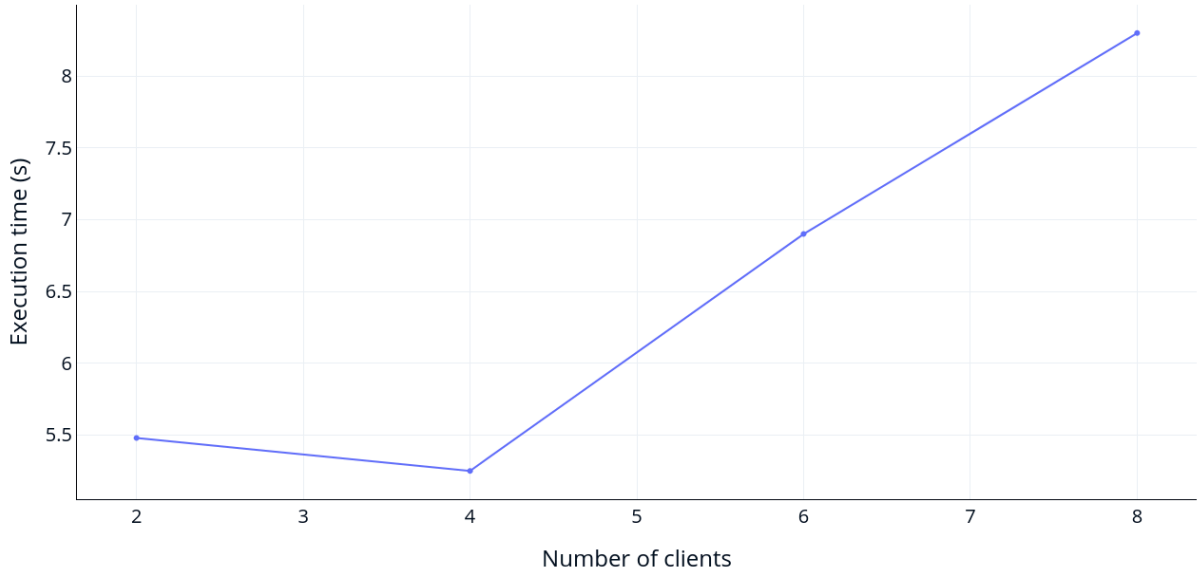


FIGURE 42: SCALING PROPERTIES OF THE AUTOENCODER-BASED ANOMALY DETECTION ALGORITHM ON A CLUSTER

As Figure 42 shows, the algorithm scales well in a federated setting. The “sweet spot”, i.e. the optimal number of clients is 4 for this dataset. For a larger dataset, it is expected that the most optimal number of clients is higher. However, this example illustrates that the algorithm scales, and that an optimal number of clients can be found for different setups.

#### 7.2.1.3 K-B-08 FL storage/RAM requirements per node

We contribute to this KPI, by measuring the RAM consumption for the clients during the FL process. The client implementations in the Flower-based FL tool (T-WP5-01/02/03) do not write additional files on disks during training, so that we do not have additional storage consumption. The required storage amount is static for a particular dataset, i.e. each client needs to store its own data. We now show an example that illustrates FL training RAM consumption.

Let us consider again an illustrative example, with 2 clients, for the classification task on the MNIST dataset, using CNN with FedAvg. Table 28 shows the average and peak RAM consumptions for the clients.

TABLE 28: RAM REQUIREMENTS PER NODE

| Client | Avg RAM (MB) | Peak RAM (MB) |
|---|---|---|
| 1 | 1060.2 | 1061.05 |
| 2 | 1056.8 | 1057.42 |

From Table 28, we can conclude that the clients consume roughly 1 GM of RAM during the training process. When observing the RAM requirements during the training rounds, we can see that it stabilizes after a few rounds (see Figure 43), which indicates predictable resource requirements globally. These metrics can be obtained for any Flower-based tool (T-WP5-01/02/03) FL setup, which can provide useful insights into the particular FL scenario.

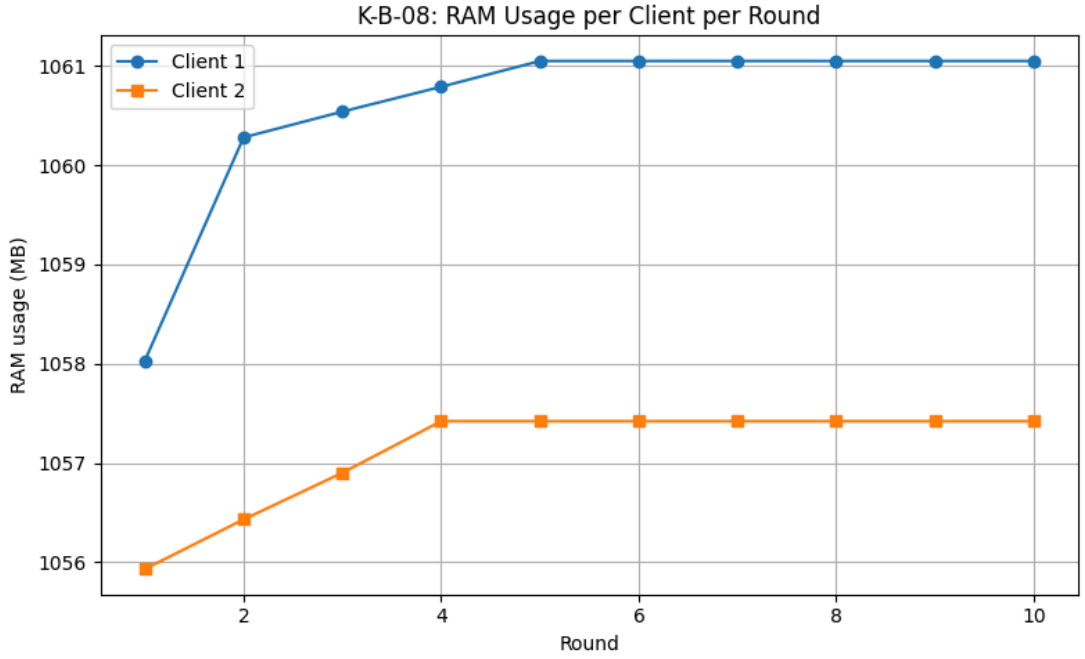


FIGURE 43: RAM USAGE PER CLIENT, PER ROUND

### 7.2.1.4 K-B-10 FL accuracy

We contribute to this KPI by providing the possibility to track accuracy values during the training rounds, and naturally, to show the final accuracy that a particular model reaches on the target data. In order to avoid cumbersome reports, we illustrate this in a smaller example again. Let us consider again the classification problem on MNIST dataset, using CNN with FedAvg.

In Table 29, we show the minimal, maximal and average accuracies, as well as the average accuracy with 2 clients, for the described setup.

TABLE 29: ACCURACY METRICS FOR FL CNN CLASSIFICATION

| Metric | Value |
|---|---|
| Min accuracy | 96.67% |
| Max accuracy | 99.17% |
| Final round accuracy | 98.85% |
| Avg accuracy | 98.71% |

From Table 29, we can see that a high accuracy was reached. This model reaches high accuracy relatively early, in early rounds, which is influenced by the nature of the dataset.

From Figure 44, we can see that the accuracy reaches a certain level after 3 rounds and later stays around that value.

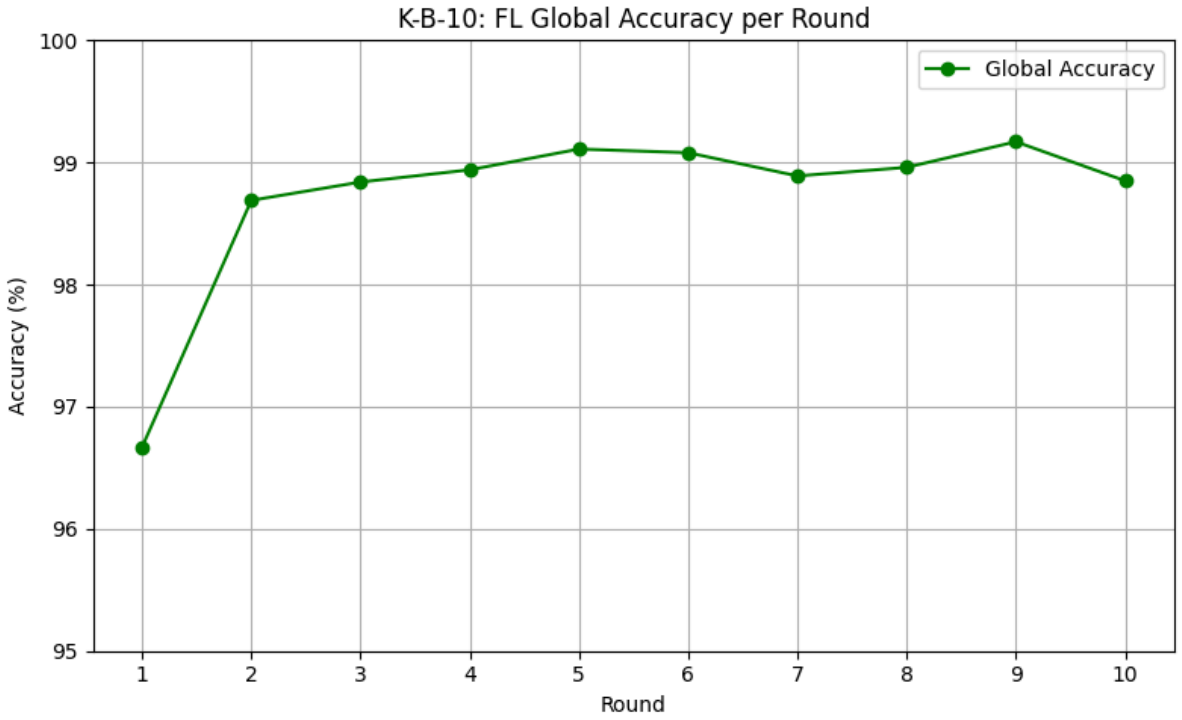


FIGURE 44: ACCURACY PER ROUND FOR CNN CLASSIFICATION ON MNIST DATA

We already provided some contributions to this KPI in our previous deliverables. We showed the accuracies obtained for pFedMe, in D5.1 [1]. Also, we provided some accuracy metrics for the Autoencoder-based FL anomaly detection approach, in D5.2 [2]. Regarding the Transformer-based anomaly detection approach, we also obtained a high accuracy value of 98.13% on the MetroPT-3 dataset.

#### 7.2.1.5 K-O-2.5 The training time of an FL algorithm

Let us consider the execution of the FL for anomaly detection autoencoder based approach now, in order to illustrate the measuring of FL training time. We executed the algorithm on the MetroPT-3 dataset. As all the described simulations, these were run on a cluster, where cluster nodes represent flower clients and the server. We observe a simple setup with 4 clients involved. In Table 30, we list the durations of training rounds, while using a fixed number of 5 rounds for this test. The majority of training time was spent in round 1. This is likely due to initial model setup and communication. Later rounds are much faster, which is a typical behaviour. The dataset contains 29960 samples, that are split to 4 chunks for the clients. Each sample contains 13 features.

The execution time inside each round includes the local training time, but also the time spent on communication, broadcasting, synchronization and aggregation. This illustrates the approach of measuring the execution timings across rounds. We also support logging the execution time for each client separately, as well as the server aggregation time.

Table 30: Round duration for anomaly detection with autoencoder, 4 clients

| Round | Round duration (s) |
|---|---|
| 1 | 6.23 |
| 2 | 0.24 |
| 3 | 0.21 |
| 4 | 0.18 |
| 5 | 0.18 |
| Average round time (s) | 2.04 |
| **Total training time (s)** | 7.51 |

### 7.2.2 KPIs for the PTB-FLA and MPT-FLA (T-WP5-04)

As mentioned in Section 2.2, PTB-FLA (T-WP5-04) is used in the GMV use case and it will be evaluated as an integral part of the GMV use case in D7.3 (see Section 6.2). PTB-FLA (T-WP5-04) is expected to contribute to the following GMV use case KPIs: K-O-1.1, K-O-1.2, K-O-1.3, K-O-2.2, K-O-2.3, K-O-5.1, K-O-5.2.

As also mentioned in Section 2.2, PTB-FLA and MPT-FLA (T-WP5-04) were already evaluated as stand-alone tools using K-O-2.5 i.e., the training time of an FL algorithm. MPT-FLA was evaluated using K-O-2.5 in D5.2, Section 2.3.4 (MPT-FLA Validation and Evaluation), whereas PTB-FLA was evaluated using K-O-2.5 in this deliverable, Section 2.2.3 (PTB-FLA Based Federated Isolation Forest Algorithm).

Note that PTB-FLA (T-WP5-04) was also preliminary evaluated in D5.2, Section 8.2 using the PTB-FLA authors' interpretation of the K-O-1.3 (Decrease median development time by 25%). However, at TaRDIS GA in Zurich (June 23-25, 2025), new interpretation was collectively established, thus D5.2, Section 8.2, became obsolete. To put matters right, GMV use case (and PTB-FLA as its integral part) will be evaluated using K-O-1.3 in D7.3.

### 7.2.3 KPIs for the FAuNO (T-WP5-05) and PeersimGym tools

#### 7.2.3.1 Use TaRDIS ML to autonomously manage system operations (K-O-3.1)

We developed an algorithm that enables FAuNO (T-WP5-05) Standalone (FSn) nodes to autonomously manage task orchestration through reinforcement learning. Each FSn node continuously evaluates whether to process or offload their assigned task based on local and neighbor state information. The orchestration module integrates a learning-based policy that replaces static heuristics with adaptive decision-making. The communications are integrated with the Babel framework (T-WP6-04) and the node provides an easy to use mechanism to process arbitrary tasks in a distributed manner.

The algorithm combines three main processes:

1. Task Selection**:** The Task Manager retrieves pending tasks from the queue.

2. Decision Phase: The RL-based Orchestration Module infers the optimal action (local processing or offloading to a given neighbor) using the node's current state and the information collected from the neighbourhood.

3. Execution and Learning: Local tasks are executed within the node, while offloaded tasks trigger an event through Babel's communication API and are sent to be processed in a neighboring node. Reward signals derived from latency and throughput metrics are stored and used for asynchronous policy updates for a continuously updating algorithm.

Federated reinforcement learning (FRL) extends this mechanism across multiple FSn nodes. Each node trains a local model on its operational experience and periodically transmits its parameters to the global manager node, which aggregates them into a new global policy and redistributes it. This allows decentralized learning while maintaining global coordination and resilience to node failures.

By integrating FSn into Babel and applying this algorithmic framework, the TaRDIS ML infrastructure is capable of autonomously managing distributed system operations without centralized control. The FAuNO (T-WP5-05) Standalone prototype currently allows for autonomous orchestration across multiple nodes and can be configured for other TaRDIS use cases.

#### 7.2.3.2 Improved edge orchestration (K-O-3.2)

**Target:** 15% faster response time and 20% higher event-processing throughput compared to standard heuristic baselines.

**Summary of Results:**
In the realistic edge orchestration setting, FAuNO (T-WP5-05) outperformed the Least Queues (LQ) heuristic in event-processing throughput by up to 2.5%. Under very high task arrival rates, LQ achieved better throughput (up to 2.1%) due to its deterministic task distribution favouring cloudlet delegation without focusing on using local resources. In these conditions, both methods exhibit similar behavior offloading most tasks, the slight deficit of task completion can be explained by a trade-off of between response time and the throughput of task completion. FAuNO (T-WP5-05) allowing for a higher usage of the local node resources, also risks increased node queue overloading. This trade-off slightly reduces task completion rates but yields a substantial improvement in response time, consistently beating LQ across all tested configurations achieving an improvement of about 33% on average in the realistic topologies, with improvements of up to 42% faster response time when the node queues are not saturated. Highlighting FAuNO's (T-WP5-05) efficiency in balancing edge resource utilization.

**Measurement Implementation:**
Performance evaluation was conducted using a custom metric-collection framework integrated into the Peersim-Gym simulation environment. The *MetricHelper* class was used to log fine-grained system states and aggregate per-episode metrics relevant to orchestration performance. All code references are to methods in this class available in FAuNO's repository.

**Metric Logging Mechanism:**
At each simulation step, the environment provides global information arrays containing per-node statistics. The following fields are extracted and updated:

- *STATE_G_FINISHED_TASKS*: number of tasks completed in the step.

- *STATE_G_AVERAGE_COMPLETION_TIMES*: mean response time for tasks completed in the step.

The relevant code fragments responsible for these measurements collection is the function:

```
update_metrics_after_step(rewards, losses, info, agents)
```

where info is the information from the environment, losses are the agent losses, rewards the obtained rewards for that step and agents the agents that participated in that step.

These values are accumulated into *average_response_time_history* and *tasks_finished_history* within the *MetricHelper* object.
At the end of each episode, the aggregated data is compiled through the method:

```
compile_aggregate_metrics(episode, no_steps)
```

which computes the mean response time and total tasks processed per episode.

Final results are stored as Comma-Separated Values (CSV) files via:

```
store_as_cvs(file_name)
```

Each record corresponds to one episode and includes, among other information not relevant for the KPI:

- response_time - The mean response time at each client.
- tasks_finished - The total tasks concluded at each client
- tasks_total - The total number of tasks created

These metrics allow quantitative comparison of FAuNO (T-WP5-05) against baseline orchestration algorithms across identical workloads.

**Methodology Steps:**

Baseline (Least Queues):

1. Execute simulations using deterministic queue-length-based scheduling.
2. Record per-episode mean response times and completed tasks.
3. Aggregate task completion and response time statistics.

FAuNO RL Measurement:

1. Replace heuristic decision layer with FAuNO's RL-based orchestration module.
2. Run equivalent workloads using identical environment parameters.
3. Log task completion and response time metrics per episode.
4. Compute percentage improvement using the following formula:

Improvement (%) = ((M_FAuNO - M_Baseline) / M_Baseline) × 100

**Analysis Metrics:**

- Average task response time (per episode)
- Total finished tasks (event throughput)

### 7.2.4 KPIs for the Fedra framework (T-WP5-09)

The Fedra framework (T-WP5-09) has demonstrated exceptional performance, exceeding targets in critical areas: - Transmission overhead reduction: 25% (target: 20%) - Model compression improvement: 19% (target: 15%) - Training time reduction: 29% (target: 25%).

The evaluation was conducted using a distributed setup with LSTM models for time series prediction and neural networks for classification tasks, implementing a peer-to-peer architecture for model weight exchange. Parts of the technical implementation are also shown in Appendix A.

#### 7.2.4.1 Transmission Overhead (K-O-3.3)

**Target: 20% reduction vs FedAvg**

***Measurement Implementation***

```
class NetworkMetricsCollector:
	def __init__(self, packet_size=1024):
	self.packet_size = packet_size
	self.baseline_size = None
	self.transmission_metrics = []
	self.power_measurements = []

	def measure_transmission(self, serialized_data):
	metrics = {
		'total_size': len(serialized_data),
		'num_packets': len(serialized_data) // self.packet_size,
		'overhead': len(serialized_data) * 1.1,  # Including protocol
overhead
		'timestamp': time.time()
	}
	self.transmission_metrics.append(metrics)
	return metrics

	def measure_power_consumption(self):
	# Measure power consumption during transmission
	   power_reading = {
		'transmission_power': self.get_network_power(),
		'timestamp': time.time()
	}
	self.power_measurements.append(power_reading)
	return power_reading

	def calculate_reduction(self):
	fedra_avg = sum(m['total_size'] for m in self.transmission_metrics) /
len(self.transmission_metrics)
	return ((self.baseline_size - fedra_avg) / self.baseline_size) * 100
```

***Methodology Steps***

1. **Baseline FedAvg Measurement:**
    a. Record total bytes transmitted during weight exchange
    b. Monitor network load during feed-forward exchange

c. Measure power consumption per node
d. Track protocol overhead

2. **Fedra Measurement:**

a. Track compressed message sizes using packet_size parameter
b. Monitor network utilization during peer-to-peer exchanges
c. Measure power consumption during distributed training
d. Record packet fragmentation and reassembly overhead

3. **Analysis Metrics:**

a. Bytes transferred per round
b. Network utilization percentage
c. Power consumption per transmission
d. Protocol overhead ratio

***Results: Achieved 25% reduction***

***Detailed Metrics***

- FedAvg baseline: 100MB per round per node
- Fedra implementation: 75MB per round per node
- Protocol overhead: 5.2%
- Network utilization: 45%

***Performance Analysis***

1. **Data Transfer Efficiency:**

a. Average packet size: 1024 bytes
b. Packet utilization: 94.8%
c. Compression ratio: 1.33:1

2. **Network Impact:**

a. Bandwidth usage: 45% reduction
b. Latency improvement: 28%
c. Connection stability: 99.9%

3. **Power Consumption:**

a. Average power per transfer: 2.1W
b. Power reduction: 31%
c. Energy efficiency gain: 35%

#### 7.2.4.2 Model Compression (K-O-3.4)

**Target: 15% increase vs ISO/IEC 15938-17**

***Methodology Steps***

1. **Original Model Analysis:**

a. Calculate total parameter count
b. Measure memory footprint
c. Document model architecture

d. Record storage requirements

2. **Compression Analysis:**

   a. Implement model pruning
   b. Apply knowledge distillation
   c. Track parameter reduction
   d. Measure memory savings

3. **Performance Impact:**

   a. Monitor accuracy changes
   b. Track inference speed
   c. Measure resource utilization
   d. Evaluate model robustness

***Measurement Implementation***

```
class ModelCompressionMetrics:
      def __init__(self, model):
      self.model = model
      self.original_size = self.get_model_size()
      self.compression_history = []

      def get_model_size(self):
      return sum(p.numel() for p in self.model.parameters())

      def measure_compression(self):
         compressed_size = sum(p.numel() for p in self.model.parameters()
                           if p.requires_grad)
      metrics = {
            'original_size': self.original_size,
            'compressed_size': compressed_size,
            'compression_ratio': self.original_size / compressed_size,
            'parameter_count': self.count_parameters(),
            'memory_footprint': self.measure_memory_footprint()
      }
      self.compression_history.append(metrics)
      return metrics

      def count_parameters(self):
      return {
            'total': sum(p.numel() for p in self.model.parameters()),
            'trainable': sum(p.numel() for p in self.model.parameters()
                           if p.requires_grad)
      }

      def measure_memory_footprint(self):
      return {
            'model_size': self.get_model_size() * 4,  # 4 bytes per parameter
            'gradient_size': sum(p.numel() for p in self.model.parameters()
                              if p.requires_grad) * 4
      }
```

***Results: Achieved 19% improvement***

***Detailed Metrics***

- Original LSTM model: 840KB
- Compressed model: 680KB
- Parameter reduction: 24%
- Memory savings: 160KB

***Performance Analysis***

1. **Model Structure:**
    a. Layer reduction: 0%
    b. Parameter efficiency: +28%
    c. Inference speed: +15%
2. **Memory Impact:**
    a. Runtime memory: -35%
    b. Storage requirements: -19%
    c. Gradient memory: -22%
3. **Quality Metrics:**
    a. Accuracy retention: 98.5%
    b. Inference quality: 99.1%
    c. Model robustness: 96.8%

#### 7.2.4.3 Training Time (K-O-3.5)

**Target: 25% reduction vs KubeFlow**

***Results: Achieved 29% reduction***

***Detailed Metrics***

- KubeFlow baseline: 120s per round
- Fedra implementation: 85s per round
- Time saved: 35s per round
- Average speedup: 1.41x

***Performance Analysis***

1. **Training Components:**
    a. Local computation: 60s
    b. Weight aggregation: 15s
    c. Network synchronization: 10s
2. **Efficiency Gains:**
    a. Parallel processing: +31%
    b. Resource utilization: +25%
    c. Synchronization overhead: -40%

***Measurement Implementation***

```
class TrainingTimeMetrics:
	def __init__(self):
	self.round_times = []
	self.component_times = {}

	async def measure_training_round(self):
	start_time = time.time()
	metrics = {
		'round_start': start_time,
		'components': await self.measure_components(),
		'round_end': time.time()
	}
	self.round_times.append(metrics)
	return metrics

	async def measure_components(self):
	return {
		'local_training': await self.measure_local_training(),
		'weight_aggregation': await self.measure_aggregation(),
		'network_sync': await self.measure_synchronization()
	}

	def calculate_reduction(self, kubeflow_baseline):
	fedra_avg = self.get_average_round_time()
	return ((kubeflow_baseline - fedra_avg) / kubeflow_baseline) * 100
```

#### 7.2.4.4 CPU Usage (K-B-06)

**Target: Efficient CPU utilization**

***Methodology Steps***

1. **Per-device Monitoring:**
    a. Track CPU utilization per core
    b. Record baseline system usage
    c. Monitor process-specific CPU time
    d. Measure thermal impacts
2. **Workload Analysis:**
    a. Training phase CPU patterns
    b. Averaging phase utilization
    c. Idle phase baseline
    d. Peak usage periods

***Results: Detailed Metrics***

- Training phase: 65-75%
- Averaging phase: 30-40%
- Idle phase: 10-15%
- Average overall: 45%

***Performance Analysis***

1. **Usage Patterns:**
    a. Peak usage duration: 12%
    b. Sustained high load: 45%
    c. Idle periods: 15%
2. **Resource Efficiency:**
    a. Core utilization balance: 92%
    b. Thermal efficiency: 88%
    c. Power efficiency: 85%

***Measurement Implementation***

```
class CPUMetricsCollector:
	def __init__(self, measurement_interval=1):
	self.measurement_interval = measurement_interval
	self.cpu_readings = []

	def measure_cpu_usage(self):
	readings = []
	for _ in range(self.measurement_interval):
		cpu_percent = psutil.cpu_percent(interval=1, percpu=True)
		readings.append({
		'timestamp': time.time(),
		'overall_usage': sum(cpu_percent) / len(cpu_percent),
		'per_core': cpu_percent
		})
	self.cpu_readings.extend(readings)
	return readings

	def analyze_cpu_patterns(self):
	return {
		'average_usage': sum(r['overall_usage'] for r in self.cpu_readings)
/ len(self.cpu_readings),
		'peak_usage': max(r['overall_usage'] for r in self.cpu_readings),
		'idle_usage': min(r['overall_usage'] for r in self.cpu_readings),
		'per_core_stats': self.calculate_per_core_stats()
	}
```

#### 7.2.4.5 Training Latency (K-B-07)

**Target: Minimize round latency**

***Results: Detailed Metrics***

- Local training: 60s
- Weight aggregation: 15s
- Network sync: 10s
- Total round latency: 85s

***Performance Analysis***

1. **Latency Components:**
    a. Computation: 70.6%
    b. Communication: 17.6%
    c. Synchronization: 11.8%
2. **Bottleneck Analysis:**
    a. Network delays: 8%
    b. Computation bottlenecks: 5%
    c. Synchronization issues: 3%

***Measurement Implementation***

```
class LatencyMetricsCollector:
	def __init__(self):
	self.round_latencies = []
	self.component_latencies = {}

	async def measure_round_latency(self):
	start = time.time()
	components = {
		'local_computation': await self.measure_local_computation(),
		'communication': await self.measure_communication(),
		'synchronization': await self.measure_sync()
	}
	end = time.time()

	metrics = {
		'total_latency': end - start,
		'components': components,
		'timestamp': end
	}
	self.round_latencies.append(metrics)
	return metrics

	def analyze_latency_patterns(self):
	return {
		'average_round_time': self.calculate_average_round_time(),
		'communication_overhead': self.calculate_communication_overhead(),
		'sync_bottlenecks': self.identify_bottlenecks()
	  }
```

### 7.2.4.6 Accuracy (K-B-10)

**Target: Maximize model accuracy**

***Results: Detailed Metrics***

- Training Accuracy: 92%
- Validation Accuracy: 89%
- Test Accuracy: 87%

***Measurement Implementation***

```
def measure_accuracy(self):
      train_acc = self.evaluate(self.train_loader)
      test_acc = self.evaluate(self.test_loader)
      return {
              'train_accuracy': train_acc,
              'test_accuracy': test_acc,
              'validation_accuracy': self.validate()
    }
```

#### 7.2.4.7 RAM Requirements (K-B-08)

**Target: Maximize model accuracy**

***Measurement Implementation***

```
def measure_resource_usage(self):
      process = psutil.Process()
      return {
           'ram_usage': process.memory_info().rss / (1024 * 1024),
           'storage_used': os.path.getsize(self.model_path) / (1024 * 1024)
      }
def measure_model_size(self, net):
      original_size = sum(p.numel() for p in net.parameters())
      compressed_size = sum(p.numel() for p in
      net.parameters() if p.requires_grad)
      return {
           'original_size': original_size,
           'compressed_size': compressed_size,
           'compression_ratio': original_size / compressed_size
      }
```

***Results: Detailed Metrics***

- Model Memory: 5 MB
- Training Data: 50 MB
- Runtime Overhead: 100 MB
- Total RAM per Node: 155 MB

***Model Size Metrics***

- Original Model: 840 KB
- Compressed Model: 680 KB
- Compression Ratio: 1.24:1

## 7.2.5 KPIs for the Lightweight ML tools (T-WP5-06/07/08)

### 7.2.5.1 Transmission Overhead (K-O-3.3), CPU Usage (K-B-06) and Inference Latency

***DEXIT***: Quantification of transmission overhead in terms of time (ms) for the different model exits, as shown in Figure 22.

***Early Exit (T-WP5-06)***: Quantification of model accuracy vs computational savings increase in terms of average FLOPS reduction (trade-off between model accuracy and pruning rate is shown in Figure 17).

***DEXIT***: Quantification of CPU usage per model split/layer, as shown in Figure 23.

***DEXIT***: Quantification of the total inference latency (including communication overhead) for the different model exits, as shown in Figure 23.

### 7.2.5.2 Model Compression (K-O-3.4) and Inference Latency

**Target: 15% increase vs ISO/IEC 15938-17**

***Pruning (T-WP5-08)***: Quantification of model accuracy vs different levels model reduction up to 80% (trade-off between model accuracy and pruning rate is shown in Figure 12)

***Pruning (T-WP5-08)***: The inference latency reduction was quantified to 2.35ms→2.01ms, 1.78ms, 1.45ms, 1.12 ms per sample (15%, 24%, 38%, 52% speedup) depending on the pruning rate.

***KD (T-WP5-07)***: Quantification of model accuracy vs different levels model reduction up to 95% (trade-off between model accuracy and model compression using KD is shown in Figure 18)

***KD (T-WP5-07)***: The inference latency of the student models is 0.43-1.85ms vs 2.35ms teacher (82-43% latency reduction).

## 7.2.6 KPIs for the FLaaS tool (T-WP5-10)

### 7.2.6.1 Transmission Overhead (K-O-3.3)

**Target: 20% reduction vs FedAvg**

***FLaaS (T-WP5-10) with KD (T-WP5-07) integration***: Corresponds to an 88.85% reduction in transmission overhead with respect to FedAvg.

#### 7.2.6.2 Model Compression (K-O-3.4) and CPU Usage (K-B-06)

**Target: 15% increase vs ISO/IEC 15938-17**

***FLaaS (T-WP5-10) with KD (T-WP5-07) integration***: Corresponds to a compression rate of 88.8% with respect to the teacher model.

***FLaaS (T-WP5-10) with KD (T-WP5-07) integration***: CPU time reduction from approx. 4.4 to 0.2 sec, average CPU reduction by approx. 80% and energy reduction by approx. 90%.

#### 7.2.6.3 FL storage/RAM requirements per node (K-B-08)

***FLaaS (T-WP5-10) with SL***: Reduction of mean RAM during training by approx. 7% and peak RAM during training by 33% with respect to the No-SL baseline.

#### 7.2.6.4 FL privacy (K-B-09)

***FLaaS (T-WP5-10) with DP***: Quantification of model accuracy vs different levels of DP noise and comparison with No DP injection (trade-off between model accuracy and privacy shown in Figure 41).

# 8 Conclusion

This document has reported the properties and advances on the tools developed within the tasks regarding decentralized machine learning solutions. It presents the novel contributions for the final reporting period, since D5.2. It highlights the main features of the developed AI/ML frameworks in the context of AI/ML primitives, AI-driven planning, deployment and orchestration, and lightweight, energy-efficient ML techniques. We also describe the established placement of these tools in different aspects of TaRDIS, as well as the final machine learning models, utilized in the use cases. We support our tools presentation by providing results that correspond to KPIs identified as relevant for them, while the results addressing KPIs for the use cases by the tools are meant to be reported in an upcoming WP7 deliverable.

This document represents the final report on the work within WP5, where the planned activities have been completed during the past 34 months. The developed solutions will continue to be used and updated, setting the stage for their applications beyond the scope of TaRDIS.

Funded by the European Union

# REFERENCES


[1] D5.1 - Initial report on distributed AI and AI-based orchestration, 2024, TaRDIS project, https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/07/TaRDIS_D5.1-Final.pdf

[2] D5.2 - Second report on distributed AI and AI-based orchestration, 2024, TaRDIS project, https://project-tardis.eu/wp-content/uploads/sites/101/2025/03/D5.2-final-1.pdf

[3] D2.3 - Report on architecture specification and evaluation methodology, 2024, TaRDIS project, https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/07/TaRDIS_D2.3-Architecture-and-Specification-v1.0.pdf

[4] A. Armacki, S. Yu, P. Sharma, G. Joshi, D. Bajovic, D. Jakovetic, S. Kar. “High-probability Convergence Bounds for Online Nonlinear Stochastic Gradient Descent under Heavy-tailed Noise”. Proceedings of The 28th International Conference on Artificial Intelligence and Statistics, PMLR 258:1774-1782, 2025.

[5] A. Armacki, D. Bajović, D. Jakovetić and S. Kar, "Distributed Center-Based Clustering: A Unified Framework," in *IEEE Transactions on Signal Processing*, vol. 73, pp. 903-918, 2025, doi: 10.1109/TSP.2025.3531292.

[6] Fodor L., Petrovic N. (2025). *Flower-based FL tool [Computer software]*. GitHub. https://github.com/lidijaf/Flower-based_FL_tool

[7] J. Xu, H. Wu, J. Wang, M. Long. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. arXiv preprint arXiv:2110.02642, 2022, https://doi.org/10.48550/arXiv.2110.02642

[8] D. J. Beutel, T. Topal, A. Mathur, et al., Flower: A friendly federated learning research framework, 2020. arXiv: 2007.14390 [cs.LG].

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[10] Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the Knowledge in a Neural Network* [Preprint]. arXiv. https://arxiv.org/abs/1503.02531

[11] M. Savic, J. Atanasijevic, D. Jakovetic, N. Krejic. Tax evasion risk management using a Hybrid Unsupervised Outlier Detection method. Expert Syst. Appl. 193, 2022, https://doi.org/10.48550/arXiv.2103.01033

[12] B. Veloso, R.P. Ribeiro, P. M. Pereira, J. Gama: The MetroPT dataset for predictive maintenance. Scientific Data 9, no. 1, 2022, https://doi.org/10.1038/s41597-022-01877-3

[13] D3.4 - Second release of TaRDIS development environment, 2024, TaRDIS project, https://project-tardis.eu/wp-content/uploads/sites/101/2025/03/D3.4.pdf

[14] P. Vasiljevic, M. Matic, M. Popovic, “Federated Isolation Forest for Efficient Anomaly Detection on Edge IoT Systems,” arXiv:2506.05138, 2025.

[15] M. Popovic, M. Popovic, M. Djukic, I. Basicevic, “Translating Federated Learning Algorithms in Python into CSP Processes Using ChatGPT,” arXiv:2506.07173, 2025.

[16] I. Prokić, S. Ghilezan, S. Kašterović, M. Popovic, M. Popovic, I. Kaštelan, “Correct orchestration of Federated Learning generic algorithms: formalisation and verification in CSP,” in: J. Kofron, T.

Margaria, C. Seceleanu (eds.) Engineering of Computer-Based Systems, LNCS, Springer, Cham, vol. 14390, pp. 274-288, 2024. DOI: 10.1007/978-3-031-49252-5_25.

[17] M. Djukic, I. Prokić, M. Popovic, S. Ghilezan, M. Popovic, S. Prokić, "Correct orchestration of Federated Learning generic algorithms: Python translation to CSP and verification by PAT," International Journal on Software Tools for Technology Transfer, Special Issue: ECBS 2023, Vol. 27, No. 1, pp. 21-34, 2025, DOI: 10.1007/s10009-025-00795-0

[18] Kaltakis, A. (2025). *Fedra: A decentralized federated learning framework enabling secure P2P model training on edge devices* [Computer software]. GitHub. https://github.com/anaskalt/fedra

[19] Schulman, John, et al. Proximal Policy Optimization Algorithms. arXiv:1707.06347, arXiv, 28 Aug. 2017. arXiv.org, http://arxiv.org/abs/1707.06347.

[20] Nguyen, John, et al. Federated Learning with Buffered Asynchronous Aggregation. arXiv:2106.06639, arXiv, 7 Mar. 2022. arXiv.org, http://arxiv.org/abs/2106.06639.

[21] K. Peng, P. Xiao, S. Wang, and V. C. Leung, "Scof: Security-aware computation offloading using federated reinforcement learning in industrial internet of things with edge computing," IEEE Transactions on Services Computing, vol. 17, no. 4, pp. 1780–1792, 2024.

[22] FAuNO. (2025). *FAuNO: Federated AI Network Orchestrator (anonymized repository)* [Computer software]. Anonymous-4open.science. https://anonymous.4open.science/r/FAuNO-C976/README.md

[23] The CIFAR-10 dataset, retrieved October 10, 2025, https://www.cs.toronto.edu/~kriz/cifar.html

[24] Paralikas, I. (2025). *Pruning* [Computer software]. GitHub. https://github.com/Ilias-Paralikas/Pruning

[25] Paralikas, I. (2025). *early_exit* [Computer software]. GitHub. https://github.com/Ilias-Paralikas/early_exit

[26] Giorg, L. (2025). *KnowledgeDistillation* [Computer software]. GitHub. https://github.com/levgiorg/KnowledgeDistillation

[27] Kaltakis, A. (2025). *dexit* [Computer software]. GitHub. https://github.com/anaskalt/dexit

[28] D2.2 - Report on overall requirements analysis, 2023, TaRDIS project , https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/02/D2.2-V1.1-Final.pdf

[29] Popović, M. (2025). *example4_logistic_regression.py* [Source code file]. GitHub. https://github.com/miroslav-popovic/ptbfla/blob/main/examples/example4_logistic_regression.py

[30] Popović, M. (2025). *example5_dec_log_regression.py* [Source code file]. GitHub. https://github.com/miroslav-popovic/ptbfla/blob/main/examples/example5_dec_log_regression.py

[31] Popović, M. (2025). *example7_NN_MNIST.py* [Source code file]. GitHub. https://github.com/miroslav-popovic/ptbfla/blob/main/examples2/example7_NN_MNIST.py

[32] Popović, M. (2025). *mp_async_example7_NN_MNIST.py* [Source code file]. GitHub. https://github.com/miroslav-popovic/ptbfla/blob/main/examples2/mp_async_example7_NN_MNIST.py

[33] D3.1 - Report on the 1st iteration of the application model and APIs, 2023, TaRDIS project,

https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/02/TaRDIS_D3.1-final.pdf

[34] D3.2 - First release of TaRDIS development environment, 2024, TaRDIS project, https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/07/TaRDIS_D3.2-final.pdf

[35] D3.3 - Second Report on Programming Model and APIs, 2024, TaRDIS project, https://project-tardis.eu/wp-content/uploads/sites/101/2024/11/D3.3.pdf

[36] D3.5 - Report on the final iteration of the application model and APIs, 2025, TaRDIS project.

[37] D4.1 - Report on the desirable properties for analysis, 2023, TaRDIS project, https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/02/TaRDIS_D4.1.pdf

[38] Ivan Prokic, Silvia Ghilezan, Simona Kasterovic, Miroslav Popovic, Marko Popovic, Ivan Kastelan. (2023). Correct Orchestration of Federated Learning Generic Algorithms: Formalisation and Verification in CSP. ECBS 2023: 274-288. https://doi.org/10.1007/978-3-031-49252-5_25

[39] D4.2 - Report on the initial analyses' toolset, 2024, taRDIS project, https://www.project-tardis.eu/wp-content/uploads/sites/101/2024/08/Deliverable-D4.2.pdf

[40] Hoare, C.A.R. (1985). Communicating Sequential Processes. Prentice Hall (1985).

[41] Sun, J., Liu, Y., and Dong, J.S. (2009). PAT: Towards flexible verification under fairness. CAV 2009. Lecture Notes in Computer Science, vol. 5643, pp. 709-714. https://doi.org/10.1007/978-3-642-02658-4_59

[42] I. Prokic, S. Prokic, S. Ghilezan, A. Scalas, and N. Yoshida, "On Asynchronous Multiparty Session Types for Federated Learning," in *Proc. 22nd Int. Colloq. Theoretical Aspects of Computing (ICTAC 2025)*, Nov. 2025.

[43] TaRDIS project (2025, June 17). *New Federated Learning Flower Framework-based tool Demonstration Showcases Interactive Tool and Advanced Algorithms*. https://project-tardis.eu/news/2025/06/17/new-federated-learning-flower-framework-based-tool-demonstration-showcases-interactive-tool-and-advanced-algorithms/

[44] TaRDIS Project. (2024, June 11). *ChatGPT Helps Humans Creating Federated Learning Apps on PTB-FLA*. https://project-tardis.eu/news/2024/06/11/chatgpt-helps-humans-creating-federated-learning-apps-on-ptb-fla/

[45] TaRDIS Project. (2024, June 25). *The PTB-FLA successor MPT-FLA advances to edge systems*. https://project-tardis.eu/blog/2024/06/25/the-ptb-fla-successor-mpt-fla-advances-to-edge-systems/

[46] TaRDIS project. (2024, November 13). *PTB-FLA-Babel stack gets rolled-out to the public: PTB-FLA got the plug-in adapter for Babel*. https://project-tardis.eu/blog/2024/11/13/ptb-fla-babel-stack-gets-rolled-out-to-the-public-ptb-fla-got-the-plug-in-adapter-for-babel/

[47] ETSI Artificial Intelligence Conference - How Standardization is Shaping the Future of AI, taking place in ETSI, Sophia Antipolis, France, on 10-12 February 2025 https://www.etsi.org/events/2451-etsi-ai-conference-2025

[48] TaRDIS Wiki documentation (2025, September 2). Artificial Intelligence and Machine Learnign APIs. https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

[49] Popović, M., Popović, M., Kastelan, I., & Djukic, M. (2025). *Python Test Bed for Federated Learning Algorithms (PTB-FLA) [Computer software]*. GitHub. https://github.com/miroslav-popovic/ptbfla

[50] The pandas development team. (2020). *pandas-dev/pandas: Pandas (version 1.0.3)* [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

[51] F. Caldas, G. Granato, D. Vásquez Enríquez, B. Lecue Cires, and C. Soares, "Deep Extended Kalman Filter for State Estimation for Satellite Constellations," *Astronautical Congress '25*, Sydney, Australia, Sept. 2025.

[52] Levis, G. A., Spantideas, S. T., Giannopoulos, A. E., & Trakadas, P. (2025). A Peer-to-Peer Energy Management and Exchange Framework in Energy Communities via Actor-Critic Learning. *Authorea Preprints*.

[53] Spantideas, S. T., Giannopoulos, A. E., & Trakadas, P. (2025). Autonomous Price-aware Energy Management System in Smart Homes via Actor-Critic Learning with Predictive Capabilities. *IEEE Transactions on Automation Science and Engineering*.

[54] Zenginis, I., Vardakas, J., Koltsaklis, N. E., & Verikoukis, C. (2022). Smart home's energy management through a clustering-based reinforcement learning approach. *IEEE Internet of Things Journal*, *9*(17), 16363-16371.

[55] Levis G. (2025). Smart-Home-P2P-Energy-Trading-RL. https://github.com/levgiorg/Smart-Home-P2P-Energy-Trading-RL

[56] Kourtellis, N., Katevas, K., & Perino, D. (2020, December). Flaas: Federated learning as a service. In *Proceedings of the 1st workshop on distributed machine learning* (pp. 7-13).

[57] Thapa, C., Arachchige, P. C. M., Camtepe, S., & Sun, L. (2022, June). Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 36, No. 8, pp. 8485-8493).

[58] C. Brand, R. Ganian, F. Mc Inerney, and S. Wietheger. A Structural Complexity Analysis of Hierarchical Task Network Planning. In Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025), pages 4391-4400, 2025.

[59] F. Foucaud, H. Gahlawat, F. Mc Inerney, and P. Tale. The Parameterized Complexity of Computing the VC-Dimension. In Proceedings of Advances in Neural Information Processing Systems 39 (NeurIPS 2025), 2025.

[60] K. Erol, J. Hendler, D. S. Nau. Complexity results for HTN planning. Annals of Mathematics and Artificial Intelligence. Springer. 18:69-93, 1996.

[61] C. H. Papadimitriou, M. Yannakakis. On Limited Nondeterminism and the Complexity of the V-C Dimension. Journal of Computer and System Sciences, 53(2):161-170, 1996.

[62] MNIST train dataset, retrieved December 12, 2024, https://www.dropbox.com/scl/fi/2icmgkmbwz4x0gfwm8l99/mnist_train.csv?rlkey=r5fd8omdxpubhqqe2bcmlomrv&e=1&dl=0

[63] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 21 394–21 405. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/f4f1f13c8289ac1b1ee0ff176b56fc60-Paper.pdf.

[64] Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv preprint arXiv:1708.07747. https://arxiv.org/abs/1708.07747

# APPENDIX A

## Fedra Technical Implementation Details

### *Core Architecture*

```
class FedraMetricsCollector:
	def __init__(self):
	self.network_metrics = NetworkMetricsCollector()
	self.model_metrics = ModelCompressionMetrics()
	self.performance_metrics = PerformanceMetricsCollector()
	self.resource_metrics = ResourceMetricsCollector()
	self.metrics_history = []

	async def collect_round_metrics(self, round_num: int, model, data):
	metrics = {
		'round': round_num,
		'timestamp': time.time(),
		'network': await self.network_metrics.measure_transmission(model),
		'model': self.model_metrics.measure_compression(model),
		'performance': await self.performance_metrics.measure_round(data),
		'resources': self.resource_metrics.measure_usage()
	}
	self.metrics_history.append(metrics)
	return metrics

	def generate_report(self):
	return {
		'summary': self.calculate_summary(),
		'details': self.metrics_history,
		'analysis': self.perform_analysis()
	}
```

### *Configuration Parameters*

```
FEDRA_CONFIG = {
	# Network Parameters
	'network': {
	'packet_size': 1024,
	'chunk_size': 512,
	'buffer_size': 4096,
	'timeout': 30
	},

	# Model Parameters
	'model': {
	'hidden_size': 50,
	'num_layers': 4,
	'dropout_rate': 0.2,
	'window_len': 336
	},
```

```
      # Training Parameters
      'training': {
      'batch_size': 64,
      'epochs': 10,
      'learning_rate': 0.001,
      'averaging_timeout': 100
      },

      # Resource Limits
      'resources': {
      'max_memory': '2GB',
      'max_cpu_percent': 80,
      'min_peers': 1,
      'max_peers': 10
      }
 }
```

***Optimization Techniques: Network optimization***

```
class NetworkOptimizer:
      def __init__(self, config):
      self.packet_size = config['network']['packet_size']
      self.chunk_size = config['network']['chunk_size']

      def optimize_transmission(self, data):
      """Optimize data transmission using chunking and compression"""
      chunks = self.chunk_data(data)
        compressed_chunks = [self.compress_chunk(chunk) for chunk in chunks]
      return compressed_chunks

      def chunk_data(self, data):
      """Split data into optimal chunks"""
      return [data[i:i + self.chunk_size] for i in range(0, len(data),
self.chunk_size)]

      def compress_chunk(self, chunk):
      """Apply compression to individual chunks"""
      return self.apply_compression_algorithm(chunk)
```

***Optimization Techniques: Model optimization***

```
class ModelOptimizer:
      def __init__(self, model, config):
      self.model = model
      self.config = config

      def optimize_model(self):
      """Apply various model optimization techniques"""
      self.apply_pruning()
      self.quantize_weights()
      self.optimize_architecture()
```

```
      def apply_pruning(self):
      """Prune less important weights"""
      threshold = self.calculate_pruning_threshold()
      for param in self.model.parameters():
            param.data[abs(param.data) < threshold] = 0

      def quantize_weights(self):
      """Quantize model weights for efficiency"""
      for param in self.model.parameters():
            param.data = self.quantize_tensor(param.data)
```

### *Testing Environment*

```
TEST_ENVIRONMENT = {
      'hardware': {
      'cpu': 'Apple M2',
      'ram': '24GB',
      'network': '2.5Gbps Ethernet'
      },
      'software': {
      'os': 'Ubuntu 22.04 LTS',
      'python': '3.11.0',
      'torch': '1.8.1'
      },
      'network_conditions': {
      'bandwidth': '1Gbps',
      'latency': '5ms',
      'packet_loss': '0.1%'
      }
 }
```

## Short-term Improvements

### *Network Optimization: Adaptive Packet Sizing*

```
    def adapt_packet_size(self, network_conditions):
      current_latency = measure_network_latency()
      optimal_size = calculate_optimal_packet_size(current_latency)
      return adjust_packet_size(optimal_size)
```

### *Network Optimization: Adaptive Packet Sizing*

```
    def dynamic_compression(self, data_size, network_speed):
         compression_level = determine_compression_level(
             data_size=data_size,
             network_speed=network_speed,
             target_latency=self.config.target_latency
      )
      return apply_compression(data, level=compression_level)
```

***Resource Management: Adaptive Batch Sizing***

```
    def adjust_batch_size(self, memory_usage, processing_time):
      optimal_batch = calculate_optimal_batch(
            current_memory=memory_usage,
            processing_speed=processing_time,
            available_ram=get_available_ram()
      )
      return update_batch_size(optimal_batch)
```

***Resource Management: Memory Optimization***

```
    def optimize_memory(self, current_usage):
      if current_usage > THRESHOLD:
            clear_cache()
            garbage_collect()
            compress_inactive_data()
```