D5.2: Second report on Distributed AI and AI-based orchestration

Revision: v.1.0

Work package	WP5
Task	T5.1, T5.2, T5.3
Due date	31/12/2024
Submission date	02/01/2025
Deliverable lead	Lidija Fodor (UNS), Dušan Jakovetić (UNS)
Version	1.0
Authors	Dušan Jakovetić (UNS), Lidija Fodor (UNS), Milica Jankov (UNS), Nemanja Petrović (UNS), Nikola Simić (UNS), Stefan Komarica (UNS), Sotiris Spantideas (NKUA), Ilias Paralikas (NKUA), Anastasios Kaltakis (NKUA), Claudia Soares (NOVA), Frederico Metelo (NOVA), Miloš Simić (UNS), Miroslav Popovic (UNS), Ivan Kaštelan (UNS), Miodrag Djukic (UNS), Pavle Vasiljevic (UNS), Simona Prokić (UNS), Ivan Prokić (UNS), Silvia Ghilezan (UNS), Alceste Scalas (DTU), Dimitra Tsigkari (TID), Manuel Pio Silva (EDP), Giovanni Granato (GMV)
Reviewers	Carlos Coutinho (CMS) Filippo Vannella (TID)
Abstract	This document represents the second 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 proposed tools are described in the context of the TaRDIS framework, defined in D2.3. This report also contains the descriptions of advances on ML modelling of the TaRDIS use cases, as well as a discussion on addressing the project objectives and important KPIs, regarding the preliminary validation approaches for the TaRDIS toolbox, defined in D7.2.
Keywords	decentralized machine learning and inference; AI/ML programming primitives, AI-driven planning, deployment and orchestration; lightweight and energy efficient ML techniques

www.project-tardis.eu

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	11/10/2024	Table of contents draft released.	UNS, NKUA, NOVA
V1.0	09/12/2024	Document ready for internal review	all authors
V1.1	22/12/2024	Document reviewed internally	CMS, 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: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

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

EXECUTIVE SUMMARY

The objective of the TaRDIS project is to develop a distributed programming toolbox that makes the development of decentralized, heterogeneous swarm applications deployed in diverse settings simpler. The main goals of work package 5 (WP5) can be described in the context of the 3 tasks it consists of: Task 5.1 developing a framework supporting artificial intelligence/machine learning (AI/ML) programming primitives, Task 5.2 providing an AI-driven planning, deployment and orchestration framework and Task 5.3 creating a library of lightweight and energy efficient ML techniques.

In Deliverable 5.1 (D5.1), an initial report on the development of distributed AI/ML primitives, lightweight ML techniques and AI-based orchestration was provided. The deliverable also contained an initial description of the positioning of the ML/AI tools in the TaRDIS framework as well as the first propositions on ML modelling approaches for the TaRDIS use cases.

This document provides an overview of WP5 contributions, for the period after D5.1 submission. It contains the descriptions of the advances on the different tasks within WP5. Regarding T5.1, the following contributions can be identified: the description and demonstration of the Flower-based federated learning (FL) tool that contains newly developed FL implementations; the introduction of split learning (SL) solutions; the illustration of advances on the Python Testbed for Federated Learning Algorithms (PTB-FLA) framework and the introduction of the MicroPython implementation of PTB-FLA (MPT-FLA) framework; and the presentation of the Fedra framework for advancing decentralised federated learning. The contributions within T5.2 include: the description of the advances on the PeersymGim environment for solving the task offloading problem with reinforcement learning; and the introduction of the Federated AI Network Orchestrator (FAuNO). The T5.3 contributions contain: the explanation of the advances on pruning, early exit and knowledge distillation lightweight ML techniques; the introduction and detailed analysis of the Decentralised Early Exit Inference Tool (DEXIT); and the description of the Communication-efficient vertical federated learning via compressed error feedback. All these tools are placed within the TaRDIS toolbox, as described in D2.3. Therefore, this document also elaborates the positioning of AI/ML tools in the TaRDIS framework, while focusing on the collaborations with activities carried out in other work packages. The advances on ML modelling are also described for all TaRDIS use cases here. Finally, a discussion on the approaches for addressing TaRDIS and WP5 objectives, as well as important key performance indicators (KPIs) is also provided, with respect to the preliminary validation approaches for the TaRDIS toolbox, reported in D7.2.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Funded by the European Union

ABBREVIATIONS

ACT	Actyx
ADMM	Alternating Direction Method of Multipliers
AI	Artificial Intelligence
API	Application Programmer Interfaces
APU	Air Production Unit
AE	Autoencoder
CFL	Centralized Federated Learning
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CSP	Communicating Sequential Processes
CSV	Comma-Separated Value
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
D	Deliverable
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDPG	Deep Deterministic Policy Gradient
DEXIT	Decentralised Early Exit Inference Tool
DFL	Decentralised Federated Learning
DR	Distributionally Robust
DRL	Deep Reinforcement Learning
EE	Early Exit
ESS	Energy Storage System
FAuNO	Federated AI Network Orchestrator
FedAvg	Federated Averaging
FL	Federated Learning
FLA	Federated Learning Algorithm
FLaaS	Federated Learning as a Service
FP8	Floating-Point 8

FP16	Floating-Point 16
FRL	Federated Reinforcement Learning
GA	Grant Agreement
GS	Ground Station
GUI	Graphical user Interface
HEMS	Home Energy Management System
HFSL	Hybrid Federated & Split Learning
HTTP	Hypertext Transfer Protocol
HUNOD	Hybrid UNsupervised Outlier Detection
HVAC	Heating, Ventilation and Air-Conditioning
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
ILP	Integer Linear Programming
I/O	Input/Output
IoT	Internet of Things
JSON	JavaScript Object Notation
KD	Knowledge Distillation
KPI	Key Performance Indicator
LEO	Lower Earth Orbit
LLM	Large Language Model
LSTM	Long Short-Term Memory
MAPE-K	Monitor, Analyse, Plan, Execute, and Knowledge
ML	Machine Learning
MLOps	Machine learning operations
MNIST	Modified National Institute of Standards and Technology
MPT-FLA	MicroPython implementation of PTB-FLA
MSE	Mean Squared Error
NAT	Network address translation
NN	Neural Network

NeuralODEs	Neural Ordinary Differential Equations
NNI	Neural network intelligence
ODTS	Orbit Determination and Time Synchronization
P2P	Peer-To-Peer
PAT	Process Analysis Toolkit
PC	Personal Computer
pFedMe	Personalized Federated learning with Moreau envelopes
PINNs	Physics-Informed Neural Networks
PPO	Proximal Policy Optimization
PTB-FLA	Python Testbed for Federated Learning Algorithms
Pub/Sub	Publish-Subscribe
RAM	Random-Access Memory
ResNet	Residual Neural Network
RL	Reinforcement learning
SGP	Simplified General Perturbations
SL	Supervised Learning
SSA	Space Situational Awareness
SSH	Secure Shell
SV	Space Vehicle
TA	Timed Automata
TDM	Time Division Multiplexing
TCP	Transmission Control Protocol
VFL	Vertical Federated Learning
vgg	Visual Geometry Group
VM	Virtual Machine
WiFi	Wireless Fidelity
WP	Work Package

Introduction

1.1 Overview

This document represents the second report regarding work package 5 (WP5) that is focused on the development of decentralised machine learning solutions. In D5.1 [1], an initial progress report was provided, aimed to comprehensively describe the ongoing work of the 3 tasks under WP5: creating a decentralised learning and inference framework supporting AI/ML primitives (Task 5.1), developing an AI-driven planning, deployment and orchestration framework (Task 5.2) and providing a library of lightweight and energy efficient ML techniques (Task 5.3). Besides that, some initial relations with tasks from other work packages were identified, which enabled an early understanding of the positioning of WP5 in the TaRDIS framework. The incipient representation of the ML modelling of the TaRDIS use cases was also provided in D5.1.

In this report, we endeavour to present the advances on each task within WP5 with respect to the reports from D5.1. At this stage, we are able to discuss the WP5 tools as components of the TaRDIS toolbox. We also provide a deeper explanation of the positioning of WP5 in the TaRDIS framework, by explaining the connections with different endeavours under various work packages. A more specific ML modelling of the TaRDIS use cases is also provided in this document, relying on the ideas from D5.1. We also discuss the main aspects of meeting the TaRDIS project objectives as well as the WP5 specific objectives. This document also contains an overview of the approaches to address the relevant Key Performance Indicators (KPIs). Finally, we provide some conclusions and ideas for next steps.

1.2 Results summary

We first present the novel contributions within Task 5.1, that concerns developing AI/ML primitives. In D5.1, we reported two different directions regarding this task: the PTB-FLA framework and FL implementations in the Flower framework. We now provide details on the advances regarding these approaches, and we also introduce two additional directions here: split learning (SL) and the Fedra framework. Besides the already reported FL algorithms implemented in the Flower framework, we discuss additional FL Flower-based algorithms implementations. The implementations are shaped into the Flower-based FL TaRDIS tool. We also provide a demonstration of this tool. Further, we discuss how ChatGPT can be of use to ease FL applications creations with the PTB-FLA framework, and we also introduce MPT-FLA, a MicroPython implementation of PTB-FLA. Additionally, we discuss distributed applications on these frameworks and aspects of formal verification of FL PTB-FLA orchestration protocols. Finally, we introduce an FL framework, Fedra. It has been developed in TaRDIS with the aim to support FL in completely decentralised, peer-to-peer (P2P) swarm systems. We describe the architecture of the framework in detail in Section 2.4.

The contributions within Task 5.2, concerning AI-driven planning, deployment and orchestration framework, include two directions. The first one is the PeersimGym environment, which is meant to address the task offloading problem by means of reinforcement learning (RL). This tool was already described in detail in D5.1. The second and novel direction is the development of the FAuNO tool. It represents a federated AI network orchestrator, developed for distributed AI systems coordination and operation enhancement. We describe the orchestration mechanism and discuss some preliminary results.

Within Task 5.3, concerning lightweight and energy efficient ML techniques, three directions were identified in D5.1: pruning, early exit of inference and knowledge distillation. These techniques were recognised and theoretically examined. Now, they have been developed and are discussed in this document. The implementations of these ML lightweight tools will be integrated in the TaRDIS toolbox. Furthermore, we introduce a new framework, the DEXIT (Decentralised Early Exit Inference Tool) framework, which aims to support the deployment of the trained Early Exit (EE) models in swarm nodes. We provide an overview of its architecture, components, workflow, as well as a description of its main features. Finally, we describe communication-efficient vertical federated learning via compressed error feedback. Beside the mentioned task specific contributions, we also present use-case specific advances in this document, as well as contributions to different aspects of TaRDIS through connections with other work packages. We also identify the relevant KPIs, while focusing on approaches to address them and provide some results for a subset of them. Finally, we discuss and identify future directions regarding TaRDIS and WP5 objectives.

1.3 Deliverable structure

The structure of this document is as follows. An introduction of the main results that represent the advances on the topics introduced in D5.1 is presented first, in Section 1. In Section 2, we present these advances regarding the framework supporting AI/ML modelling primitives with detailed progress descriptions of the following directions under T5.1: The Flower-base FL tool, split learning, the PTB-FLA and MPT-FLA frameworks and the Fedra: Advancing decentralised federated learning framework. In Section 3, we discuss the advances related to the AI-driven planning, deployment and orchestration framework, focusing on the thorough explanations of the improvements on the PeersymGim environment for solving the task offloading problem with reinforcement learning and the FAuNO: Federated AI network orchestrator tool. Section 4 is dedicated to progress description regarding lightweight, energy-efficient ML techniques, including pruning, early exit and knowledge distillation. This section also provides a detailed analysis of the DEXIT framework and finally, it describes a communication-efficient vertical federated learning via compressed error feedback. Section 5 presents the positioning of AI/ML tools in the TaRDIS framework, by examining the connections and collaborations with tasks from different work packages. Section 6 provides a detailed overview of the current state of the art regarding ML modelling of the four TaRDIS use cases, while Section 7 discusses the approaches to meet the project and work package specific objectives. Section 8 is dedicated for examining the contributions to relevant KPIs. Finally, Section 9 concludes this report, by summarizing the most important aspects.

2 Advances on framework supporting AI/ML modelling primitives

In this section, we describe the contributions made regarding the development of AI/ML programming primitives. In D5.1, we discussed two frameworks of interest: the Flower and the PTB-FLA frameworks. In this section, we also provide an overview of an ongoing work regarding split learning, by highlighting the planned activities and some preliminary considerations. In D5.1, we described the PTB-FLA development paradigm and presented an implementation example. Now, we expand the topic further here and discuss the usefulness of ChatGPT in creating PTB-FLA implementations and introduce a MycroPython implementation of the framework, MPT-FLA. We also list the relevant publications on these topics. Finally, we introduce a new framework, Fedra, for advancing decentralized FL, and describe the architecture and features of the Fedra framework in detail. We introduced personalised and clustered FL implementations in the Flower framework. We also describe the advances in the context of the Flower framework in this section, by introducing and demonstrating the Flower-based FL tool and discussing new Flower-based FL implementations, namely distributionally robust FL and anomaly detection. In addition to the above efforts presented ahead in detail in the current section, we also report here briefly on methodological T5.1 advances in the context of clustered and robust distributed learning [2], [3]. Namely, theoretical advances related to clustered learning have been developed in [2], where we investigate the influence of centre initialization on performance of distributed gradient-based clustering algorithms. We demonstrate the resilience to initialization effects for these methods and propose a novel distributed centre initialization scheme. For more details on the results, we refer to [2]. In the context of robust learning, we also present recent theoretical advances on heavy-tailed noise in distributed estimation [3], where we introduce a distributed estimation algorithm in an environment with heavy-tailed observation and communication noises. We present results on convergence and asymptotic performance, as well as on trade-offs between system noises and the underlying network topology. For more details, we refer to [3]. The incorporation of methodologies developed in [2,3] in the Flower-based FL tool will be considered in the final year of the project.

2.1 The Flower-based FL tool

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) have been defined within the TaRDIS architecture definition in D2.3 [4], where the descriptions have been supported by a set of diagrams defining the tools architectures, workflows and behaviours. The tools provide federated machine learning solutions and enable model training. The envisioned functioning was presented by mock-ups in D3.2 [5]. We now present the Flower-based FL tool, that represents a synthesis of the mentioned 3 tools, as they are envisioned to work in synergy. During the previous period, we mainly focused on the development of the Flower-based FL model training tool (T-WP5-01), where we extended the list of FL model training algorithms. The list of these algorithms will be expanded further, according to the needs. Additionally, we developed some initial inference capabilities (T-WP5-03), that will be expanded in the upcoming period. We also plan to start working on the preprocessing approaches (T-WP5-02) during the finishing phase of the project. We now demonstrate the functioning and usage of the Flower-based FL tool, that supports the developer during the process of setting the training up, so that no FL expertise is needed in order to train a model. We also discuss the novel FL-based Flower implementations here, which expand the list of already reported algorithms, i.e., federated averaging, personalised and clustered FL. The algorithms that we consider here are distributionally robust FL and anomaly detection in the Flower framework.

2.1.1 The Flower-based FL tool demonstration

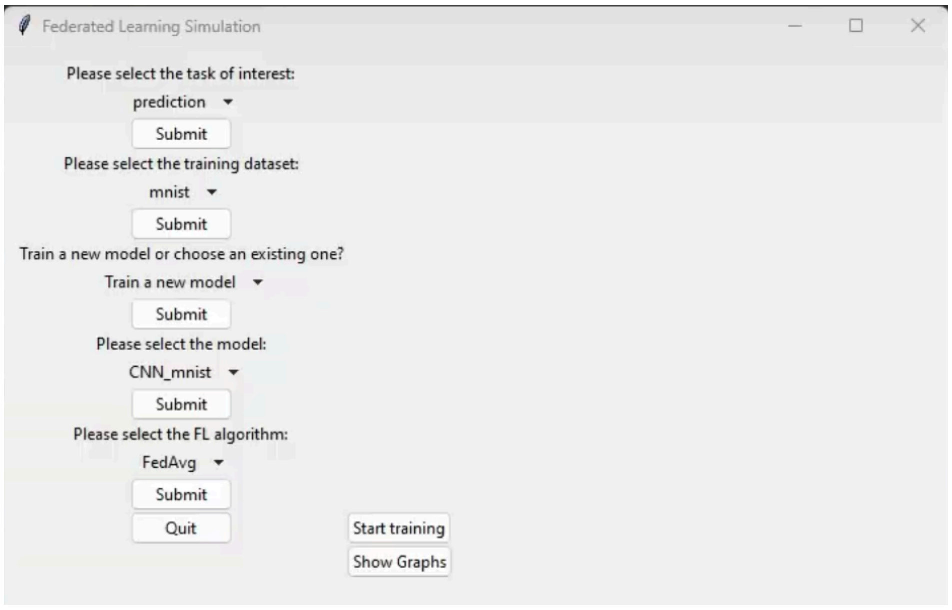
One of our main focuses is the development of an application that uses the Flower framework for FL. This application enhances the accessibility and user-friendliness of FL techniques, particularly for non-expert users. This tool covers the Flower-based tools T-WP5-01,T-WP5-02 and T-WP5-03, introduced in D2.3 [4]. It meets a set of requirements (from D2.2 [6]), that, among others, include providing a list of FL algorithms and supporting diverse ML algorithms in decentralised frameworks.

The primary innovation lies in the intuitive interface and streamlined workflow, which simplifies the complex process of distributed model training. It is in line with TaRDIS candidate applications 4.3 - 4.5 proposed in D3.2 [5] for the TaRDIS Toolbox. It provides both command-line usage, as well as a graphical user interface (GUI).

The application offers flexibility in model selection and initialization. Users can either select a new model, such as a convolutional neural network, or load pre-existing models into the system. This feature allows for versatility in addressing various machine learning tasks based on the specific requirements of the user.

Building on the flexibility of our application, we have implemented a comprehensive system for customising the learning process while applying FL. Users have the ability to tune training parameters, allowing for precise control over the learning environment. These parameters encompass various aspects of the FL setup, including the number of training rounds, the total client pool size, and the batch size for local computations. Additionally, users can specify the number of classes in their classification task, adjust the client participation rates for both model fitting and evaluation phases, and modify key hyperparameters of the optimization process. This level of customization extends to learning rate settings, momentum values for the optimizer, and the number of local epochs performed by each client. By providing this type of control over the training process, our application empowers users to optimise their FL models for diverse scenarios and dataset characteristics, facilitating more effective and efficient model development across a variety of use cases.

As already mentioned, the user can interact with the tool by a command-line interface, or by using a GUI. We illustrate the usage of the GUI, as it is more convenient for most of the users. First, the user needs to select the tasks of interest, for instance prediction, forecasting, anomaly detection etc. Then, the dataset needs to be selected. The user can then choose between two options: training a completely new model or using an existing, pretrained model. Finally, the user can select the model, as Convolutional Neural Network (CNN) for example, and an FL algorithm, as Federated Averaging (FedAvg), Personalized Federated learning with Moreau envelopes (pFedMe), etc. The system guides the user through these steps, and provides a safe environment, as the available choices depend on the previous selections. This way, the tool will not allow the user to set up a training that does not apply for the selected options. We plan to work even further on the possibility of checking the applicability of the created setup before starting the training, in the future. The process of selecting the described options is presented in Figure 1.



Federated Learning Simulation

Please select the task of interest: prediction

Please select the training dataset: mnist

Train a new model or choose an existing one? Train a new model

Please select the model: CNN_mnist

Please select the FL algorithm: FedAvg

Figure 1: *Setting up the training in the Flower-based FL training tool.*

As the training proceeds, the tool produces results, in terms of loss and accuracy. However, when the training is finished, it offers graphical representations of the results. For instance, for the setup in Figure 1, the tool produces the outcomes shown in Figure 2. The graphs show the accuracy and loss plotted over the training rounds. The horizontal axis displays the training rounds and the vertical axis shows the accuracy, on the left, and the loss, on the right. These outcomes may differ for different training scenarios, as we may have different graphs to show, depending on the selected model and algorithm.

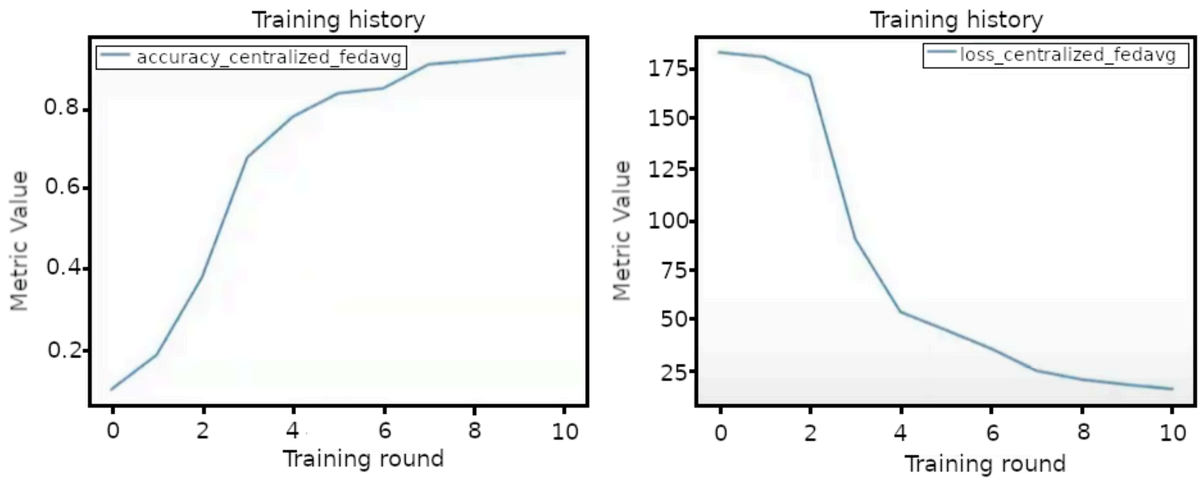


Figure 2: *An example of the output of training for the Flower-based FL model training tool.*

A key aspect of our implementation is its focus on ease of use. By abstracting away much of the complexity inherent in FL setups, our app makes the Flower framework more accessible to a broader audience. This approach eases access to FL technologies, enabling researchers and practitioners without deep expertise in distributed systems or machine learning to leverage these powerful tools.

2.1.2 Distributionally robust FL

The Flower-based FL model training tool (T-WP5-01) under the Flower-based FL tool aims to provide a list of implemented FL algorithms and support diverse ML algorithms in decentralised frameworks. Therefore, we propose a novel FL algorithm, named Distributionally Robust (DR) FL, that is meant to be implemented in the Flower framework.

The aim of this approach is to find a common ML model that performs well on any client's data set. This means that we formulate the problem so that we strive to find an ML model that performs well in the worst case over a "region" of data distributions.

More formally, assume that each client i, $i = 1,..., N$, holds a local loss function $F_i: R^d \to R$. For example, F_i may be the empirical loss associated with the client i's local data set sampled from a distribution D_i. The distributionally robust FL aims to find a model (e.g., weights of a neural network) $x \in R^d$ that performs well on the data coming from the distribution from any client or any convex combination of such distributions; see, e.g., [7]. A possible mathematical formulation to tackle this problem is the following:

$$minimize_{x \in R^d}\, max_{\lambda \in \Delta} \sum_{i=1}^{N} \lambda_i F_i(x).$$

Here, $\Delta = \{\lambda \in R^N : \lambda_i \geq 0\, for\, all\, i,\ \sum_{i=1}^{N} \lambda_i = 1\}$ is the N-dimensional probability simplex. It is possible to devise distributed and federated methods to tackle problems of the above form, e.g., [8,9]. However, a major computational challenge here is that the dimensionality of the uncertainty set Δ scales linearly with the number of clients. In order to tackle this challenge, we propose a method that simultaneously learns a model x and clusters clients into K groups according to similarity of their data distributions, where $K << N$ is a tuning parameter. It is often reasonable to assume that the clients may be partitioned into groups such that their data distributions are mutually identical or similar. For example, the client groups may correspond to clients in geographical proximity, or to devices of the same type. In our formulation, the clustering of clients, i.e., grouping of clients according to the similarity of their distributions, is assumed to be unknown beforehand.

The algorithm can be described by the following pseudo-code:

1. At a global round $t = 0, ..., T - 1$, the server broadcasts the global model x_t to each client i
2. Each client calculates its local loss: $F_i := F_i(x_t)$
3. Each client calculates a stochastic gradient $g_{i,t}$ of function F_i at argument x_t, e.g., by by using a mini-batch of data available at client *i*
4. Each clients sends the pair $(F_i, g_{i,t})$ to the server
5. The server clusters the vectors $g_{1,t}, ..., g_{N,t}$ into K clusters C_k, $k = 1, ..., K$
6. The server calculates $f_k := \frac{1}{|C_k|} \Sigma_{j \in C_k} F_j$ and finds $f_{j^*} := max_{k=1,...,K} f_k$, that is, the server finds the average cost across each cluster, and determines the cluster $j^* = argmax_k f_k$ that has the maximal cost value. Here, $|C_k|$ denotes the cardinality of set C_k.

7. The server calculates $g_{final,t} = \frac{1}{|C_{j^*}|}\Sigma_{j\epsilon C_{j^*}} g_{j,t}$, that is, the server calculates as its search direction the average of the gradients of all clients that belong to the "currently hardest" cluster, i.e., the cluster with the highest associated cost function
8. The server updates the model as $x_{t+1} = x_t - \alpha_t g_{final,t}$, where $\alpha_t > 0$ is the step-size (learning rate).

Intuitively, the algorithm above simultaneously learns the similarity of the data distributions across different clients and fits the model that is aimed to perform well on the data that comes from any client. Setting the parameter *K* (number of clusters) to one, the algorithm above reduces to the standard FedAvg method. On the other hand, setting *K*=*N* reduces to a (sub)gradient descent on the function $max_{\lambda\epsilon\Delta} \sum_{i=1}^{N} \lambda_i F_i(x)$. A more detailed performance evaluation of the algorithm, both analytical and numerical, is left for future work.

The algorithm is being implemented in the Flower framework and is currently under testing. From the implementational perspective, there are some challenges that arise here. First, the aggregation of the results from the clients needs to be customised. This is necessary, as the default approaches do not fit the needs of the algorithm. By default, in the simplest case, the Flower framework expects to collect the local parameters from the clients and aggregate them using a predefined strategy. Flower provides an approach for this, called Strategy abstraction. There is a variety of provided built-in strategies available in Flower, for instance FedAvg. However, Flower enables building a custom Strategy, i.e., a custom FL algorithm on the server side. A strategy needs to define some methods, derived from the abstract base class Strategy. One of these methods is *aggregate_fit*, which is responsible for aggregating the results returned by the clients. Here, we can implement our clustering approach (see step 5 in the algorithm above), by using the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [10], for instance.

However, the most challenging part of the implementation is to ensure that the clients send the weights, the losses and the gradients to the server. This seems straightforward, but when looking deeper into the way how Flower passes the results from the clients to the server, it can be seen that a custom approach needs to be built. By default, each client returns the updated parameters, or the gradients, as a list of arrays, by implementing the appropriate method for getting the parameters. The server receives a special type of object for each client, called FitRes. We define a way to serialise the parameters, the losses and the gradient to FitRes on the clients. We use the Client [11] class instead of the NumPy client, as it is highly customizable, so that we can apply our implemented mechanisms. Listing 1 shows the method for getting the parameters on the client. It gets the updated parameters, the loss and the gradient, and serializes them to the response object.

```
def get_parameters(self, ins: GetParametersIns) -> GetParametersRes:
     print(f"[Client {self.cid}] get_parameters")
     # Get parameters as a list of NumPy ndarray's
     arrays: List[np.ndarray] = get_parameters(self.net)
     #Get the gradient and the loss
     gradient: np.ndarray = get_grad()
     loss = get_loss()
     # Serialize everything together, build and return response
     parameters = ndarrays_to_parameters(arrays, gradient, loss)
     status = Status(code=Code.OK, message="Success")
     return GetParametersRes(status=status, parameters=parameters)
```

Listing 1: The custom get_parameters method on clients.

The implementation of the DR FL algorithm is currently under testing. The next step is a detailed evaluation of the implementation, regarding accuracy. Also, the scalability is an aspect that will be evaluated by running simulations on a cluster environment. Finally, the algorithm could be possibly applied to some real-data scenarios.

2.1.3 Anomaly detection

The development of an algorithm for anomaly (outlier) detection was inspired by the Actyx (ACT) use case (see ahead Section 6). Since the ground truth labels are unavailable in the context of this use case, the most suitable approach for addressing the problem relies on unsupervised learning techniques. The implementation integrates two distinct yet complementary machine learning methodologies: clustering using the K-means algorithm and representational learning through autoencoders. When developing our method, we build upon the HUNOD (Hybrid UNsupervised Outlier Detection) method [12]. With respect to [12], we are working on providing several innovations. First, we aim to make the method robust to inexact or noisy label information–a scenario highly relevant for factory cases wherein labels may be obtained by automated methods subject to errors in the absence of a human labeller. Second, we aim to generalize the method to federated learning settings.

The core of the developed anomaly detection method focuses on using an autoencoder for outlier detection. In our approach, the autoencoder is trained using a subset of instances from the dataset, selected based on predefined knowledge that they represent normal (non-anomalous) instances. A significant advantage of the developed model is its capability to perform outlier detection without requiring true positive values during training.

In the developed method, the TensorFlow framework [13] is employed for training the autoencoder and enabling it to perform outlier detection inference. The entire procedure is detailed in Algorithm 2 of reference [12]. The autoencoder is constructed using the TensorFlow sequential neural network model, with the specified number of hidden layers added. The values for the regularisation hyperparameters are set to $\alpha = 0.8$ and $\lambda = 0.1$, as proposed in the original implementation. The loss function is optimised using the Adam optimization algorithm [14] in a given number of epochs e and batch size b [12]. The default values of the autoencoder hyperparameters are $e = 200$ and $b = 32.$ The activation function is set to *ReLU*, $\sigma(x) = max(0, x)$ [12].

All outliers detected by the autoencoder are then verified against those identified by K-means algorithms utilised in our approach. We used two different clustering approaches. First, we utilized the K-means clustering algorithm ([15], [16]), as proposed in the paper [12]. We relied on the K-means algorithm from the scikit-learn Python library [17].

When the clustering is finished, the outlier detection can be done by observing small clusters that are distant from large ones. Further details on the base algorithm can be found in reference [12]. Additionally, we aimed to explore the use of non-fixed K values, which was not proposed in the original paper. Therefore, for all integers within the desired range, we applied the clustering algorithm and subsequently determined the optimal K using the Silhouette score. This adds an additional element for comparing the outlier detection approach by the autoencoder and by the K-means algorithms. The optimal clustering configuration was determined by using non-anomalous data by Silhouette score. We then obtained the distances of the test data points from the centroids, in order to mark the points that are significantly distant from all the centroids, as outliers. We used a threshold value for the distance, based on the 95th percentile of distances calculated from the training set.

This corresponds to a centralized learning approach, where an autoencoder model was used for anomaly detection in the ACT use case, as detailed in Section 6, where we show the results of the experiments. This centralized learning workflow has been consequently adapted into an DL paradigm using the Flower framework and the FedAvg algorithm.

The distributed setup involves distributing the anomaly detection task across multiple clients, each simulating a decentralized node. The Flower framework is employed to coordinate the interactions between the clients and the server. Clients train their local autoencoder models on their respective datasets while ensuring that their data remains private throughout the process. Each client computes and updates its local anomaly detection threshold based on the training phase. This threshold is then stored locally and used in anomaly classification tasks. At the server, the model parameters are aggregated using the FedAvg algorithm to update the global model. Additionally, the server aggregates the local thresholds received from the clients to compute a global threshold for evaluation purposes.

To assess the effectiveness of the FL approach, future evaluations will analyse performance using metrics such as accuracy, precision, recall, and F1 score. These evaluations will focus on both the global model using the aggregated thresholds and the aggregated metrics derived from local models and thresholds. Moreover, the results will be compared against the evaluation of the centralized autoencoder model to provide a comprehensive view of the performance trade-offs between centralized and FL approaches.

2.2 Split learning

In this section, we discuss a new direction of interest for research in the context of developing support for AI/ML primitives, named split learning (SL). SL represents an approach that allows different portions of an ML model to be collaboratively trained on different workers in a learning framework. In split federated learning, a model is partitioned into (at least) two segments: one that resides on the local device and another that is centralised. During training, only the intermediate activations (or gradients) between these split layers are transmitted between the device and the server. This process inherently involves communication costs, which include both the bandwidth required and the energy consumed during data transmission.

To enhance energy efficiency and reduce transmission costs, we propose applying various quantization techniques between the split layers of the model. Specifically, we will investigate some of the following techniques: deterministic and stochastic binarization, 2-bit quantization, and Floating-point 8 (FP8) quantization methods. The impact of these techniques will be evaluated considering problems related to some of the following datasets: CIFAR-10 [18], Modified National Institute of Standards and Technology (MNIST) [19] and IMDB [20]. Our goal is to ensure that the approaches involving quantization maintain performance levels comparable to the full precision (32-bit) models while achieving significant reductions in transmission costs. To facilitate this research, we will utilise and extend an existing open-source codebase from the GitHub repository associated with the paper [21].

Currently, we conducted initial experiments on the CIFAR-10 dataset in the context of the generalized split federated learning setup, whose schema is presented in Figure 3. We considered 8 client devices and 4 groups (each communicating with two client devices) connected to the main server. On the client side, the model consists of two convolutional layers with 32 and 64 kernels, each of size 3×3, followed by a Max Pooling layer. The output of the Max Pooling layer is sent to the main server, where a linear layer with 128 nodes and the output layer are deployed.

Funded by the European Union

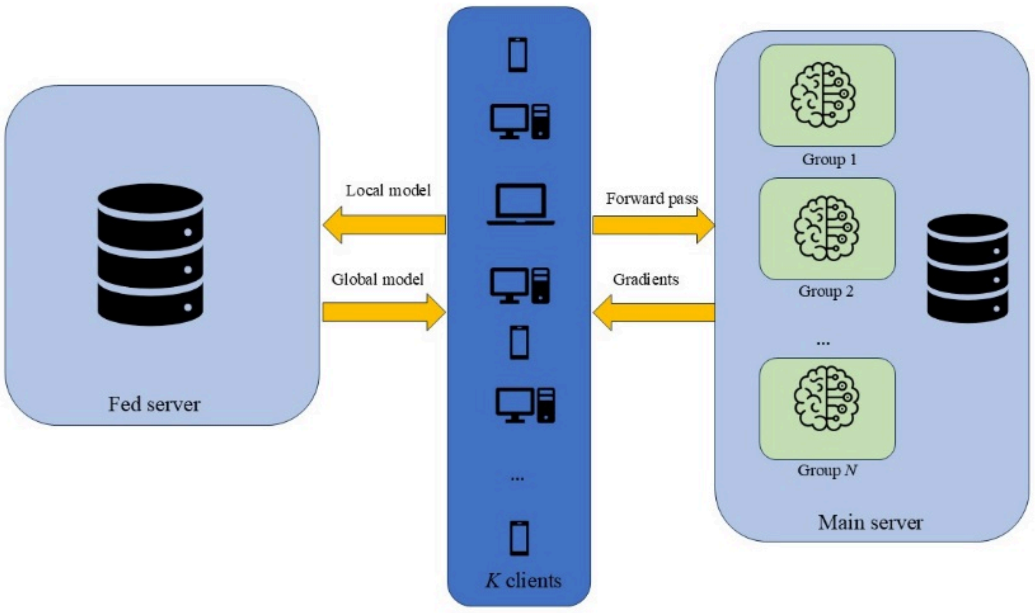


Figure 3*: Split federated learning schema.*

The comparison of classification accuracy between the full-precision 32-bit model and the FP8 model is presented in Figure 4.

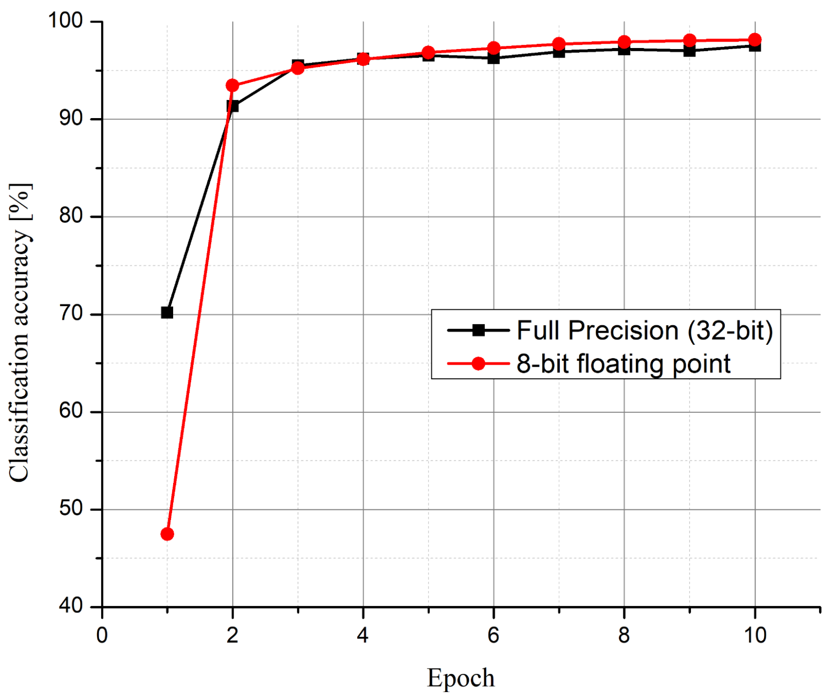


Figure 4*: Performance of the full-precision and FP8 model.*

As expected, applying FP8 arithmetic to the output of the split layer provides performance comparable to the full-precision model. A possible scalar quantization solution with the highest compression ratio is binarization. However, deterministic binarization did not provide satisfactory performance, and the achieved classification accuracy was very low. This motivated us to explore stochastic binarization. Initially, applying stochastic binarization to the split layer of the generalized split federated learning algorithm led to an accuracy of 76%. In the next phase of our research, we will further explore other averaging techniques at the federated server, as well as alternative quantization approaches.

2.3 PTB-FLA AND MPT-FLA

PTB-FLA, identified as T-WP5-04 in D2.3 [4], is a framework that comes to the rescue for developers of FL algorithms. Within the vast number of FL frameworks, it provides a development and test environment for the development of FL algorithms that is lightweight and easy to install and program by both nonprofessional programmers and Large Language Models (LLMs) like ChatGPT. The blog published on TaRDIS website explored how ChatGPT can help humans create federated learning apps on PTB-FLA [22].

In another blog post, the TaRDIS research team provided a brief overview of the MicroPython implementation MPT-FLA that aims to take what its predecessor did one step further – to the local network [23].

The video demonstration was made to show the validation of MPT-FLA [24]. In this video demonstration, the Centralised Averaging app runs on four nodes. A server node is on the PC. The client nodes are on two Raspberry Pi Pico W boards and a Hussarion ROSbot2 PRO robot.

In the beginning of the video, a node 0 (master node) is started on a PC. The two Raspberry Pi Pico W boards are then powered on and ROSbot sets its navigation goal. The node on ROSbot is started over the secure shell (SSH) communication. When the robot reaches its goal, the application terminates with the expected results.

2.3.1 PUBLISHED RESULTS AND VIDEO DEMONSTRATION

More details about the PTB-FLA (T-WP5-04) and MPT-FLA can be found in the following published manuscripts:

- The paper that introduced the PTB-FLA system architecture and its validation on the first three examples (Federated Map, Centralised and Decentralised Data Averaging) was published in the proceedings of the ZINC 2023 conference [25].
- In the proceedings of the ECBS 2023 conference, a development paradigm was proposed for PTB-FLA that consists of four steps: 1) the referent sequential code, 2) the federated sequential code, 3) the federated sequential code with callbacks, and 4) the PTB-FLA code [26].
- A formal verification of the correctness of two federated learning algorithms using Communicating Sequential Processes (CSP) was published in the proceedings of the ECBS 2023 conference [27].
- Developing code is lately supported by AI tools. The TaRDIS research team explored how ChatGPT can help implement federated learning algorithms in PTB-FLA. The findings were published in the proceedings of the TELFOR 2023 conference [28].
- An adaptation of the PTB-FLA development paradigm for ChatGPT was published in the Computer Science and Information Systems journal in September 2024 [29].
- A future paper will introduce the MicroPython implementation of PTB-FLA, named MPT-FLA. At the time of writing this report, the paper was available in preprint [30].
- Finally, a future paper will introduce a work in progress on formal verification of federated learning orchestration protocols on satellites. At the time of writing this report, the paper was available in preprint [31].

2.3.2 PTB-FLA Development Paradigm Adaptation for ChatGPT

In this section, we provide a short digest of the research that was published in the open access journal ComSIS [32], which comprises the problem formulation and its solution, as well as this research achievements.

Problem Formulation and Its Solution. The problem that authors tried to solve, was how to adapt the PTB-FLA development paradigm used by humans for ChatGPT in order to (i) minimise the human labour by delegating a part of the job to ChatGPT, and at the same time to (ii) minimise the ChatGPT input by creating minimal contexts, because commercial AI services are charged according to the input they processed.

The problem was solved experimentally by using the simple iterative development process: the authors used the text (guidelines) of the original development paradigm to create the initial ChatGPT context, and then in each iteration they used feedback from ChatGPT to manually modify the context i.e., to adapt the paradigm, aiming towards the minimal context without redundant parts. In each iteration, the current paradigm (given in the current context) was also validated, because it had to produce the output PTB-FLA code that is semantically equivalent to the input sequential code. The application code that was used for this purpose is the logistic regression case study.

Research Achievements. As a continuation of [26] and [28], the authors firstly adapted the four-phases development paradigm originally devised for humans to guide ChatGPT to successfully develop the same algorithms as in [26]. Secondly, they adapted the four-phases paradigm into a two-phases development paradigm. As the first phase is always done by humans, the two-phases paradigm is rather close to the ideal solution where developers in the second phase immediately get the complete target PTB-FLA code.

It should be emphasised that both adapted paradigms (the four-phases and the two-phase) are original contributions, and both should be treated unbiasedly as both have their own strengths and weaknesses. The four-phases paradigm requires more human labour but provides better traceability (somewhat like grey box testing), whereas the two-phase paradigm requires minimal (almost no) labour but is less traceable (being based on the black box approach) and therefore riskier.

Both development paradigms were experimentally validated. In the experimental validation, the well-known GPT-3.5 model was used. Authors reported on the adapted development paradigms performance in terms of human labour and size of ChatGPT context needed to develop the logistic regression PTB-FLA code, see the results in section 8.2 of this document. For more details on this research see [32].

2.3.3 MPT-FLA Distributed Launcher

As the validation of MPT-FLA expanded to include more devices, the process became increasingly time-consuming, prone to human error, and difficult to test repeatedly. To address these challenges, the Distributed Launcher was developed as a solution.

The Distributed Launcher for MPT-FLA is a distributed application that simplifies and automates the execution of applications written utilising the MPT-FLA framework. In addition to this, it provides a way to reliably and repeatedly test and measure the execution time of federated learning applications without the need to manually start every instance. Execution time measurement results are exportable to Comma-Separated Value (CSV) format, which is convenient for further analysis.

The Distributed Launcher comprises three main components: a user-friendly GUI for ease of use, a master node that coordinates the application's internal processes, and several agent nodes that execute the actions dictated by the master node. The architecture of the MPT-FLA Distributed Launcher is shown in Figure 5.

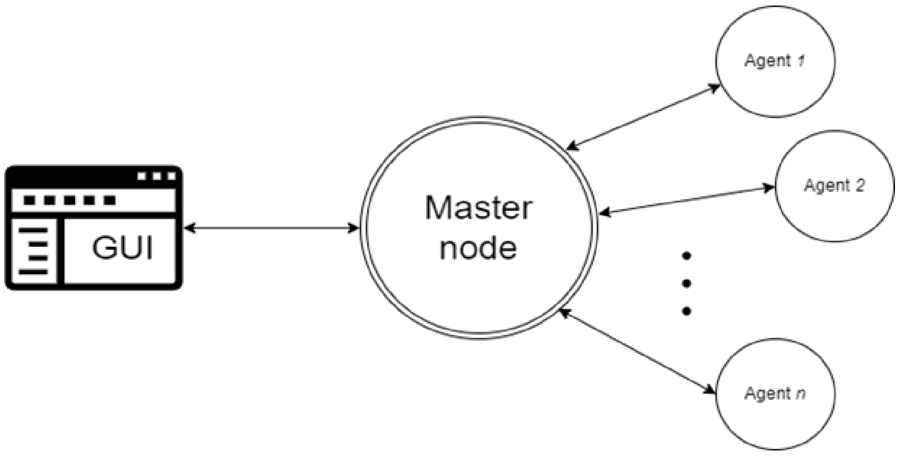


Figure 5*: The MPT-FLA Distributed Launcher Architecture.*

Functionalities that the distributed launcher encompasses are the following: agent node discovery, starting of multiple MPT-FLA instances on *n* hosts from a single point, MPT-FLA application execution time measurement, and storage and analysis of measurement-related data.

Agent discovery is facilitated through a simple multicast-based discovery protocol, while communication between nodes occurs over standard transmission control protocol (TCP), utilising JavaScript Object Notation (JSON) as the data format. To store and present measurement data in a more structured way, the subsequent behaviour modelling abstractions have been introduced: a *session* and a *batch*. *Session* is defined as executing an MPT-FLA based application, given the number of batches with varying application parameters, while the term *batch* corresponds to executing an MPT-FLA based application *m* times with the same application parameters. Previously mentioned abstractions improved data integrity and structure data in a way that is convenient for analysis. Another strength of the Distributed Launcher is its portability and lightweight nature of agent nodes, using only the abstractions provided in the Python standard library. The distributed launcher's GUI is implemented as a web application that communicates with the master node via Hypertext Transfer Protocol (HTTP), requiring only a web browser for access. It is split into three sections: launching, measurement, and analysis, reflective of their uses.

Overall, the Distributed Launcher for MPT-FLA improves the developer experience, making testing and evaluation smoother and less mistake-prone, automating some of the labour required when testing MPT-FLA based FL applications on multiple devices.

The Distributed Launcher for MPT-FLA was developed by Pavle Vasiljevic in his B.Sc. thesis (available in Serbian [33]), mentored by Miroslav Popovic, and is publicly available in the distributed Launcher GitHub repository [34], which is also accessible over a link in the “ptbfla” GitHub repository [35].

2.3.4 MPT-FLA Validation and Evaluation

The MPT-FLA framework was experimentally validated on a WiFi network, consisting of one WiFi router Belkin F5D7234-4, two Raspberry Pi Pico W boards, and one PC, by using the four adapted algorithm examples originally developed for the PTB-FLA framework: federated map, centralised data averaging, decentralised data averaging which were introduced in [25] and Orbit Determination and Time Synchronization (ODTS) introduced in [30].

The MPT-FLA successfully passed this experimental validation because, as expected, the adapted algorithms produced the same numerical results as the originals, and this was the sole goal of this experiment validation. However, two WiFi related issues appeared during the experiments that might compromise its validity: (1) repetitive WiFi connecting attempts by Pico boards were taking progressively more time, as remedy pauses were made between the individual sessions, and (2) under strong WiFi interferences, especially in case of overlapped networks, connections could be broken [30], therefore the experiments were conducted in the laboratory.

The MPT-FLA framework was experimentally evaluated using the Distributed Launcher for MPT-FLA on the wired network, consisting of two D-LINK DGS-1016D routers and 18 PCs (i7 6700, 16 GB), by measuring the mean execution time (mt) versus the number of nodes (n), for the same four adapted examples, as follows. For each example, a separate session was conducted. Each session comprises 5 batches of measurements, and each batch comprises 30 measurements. For each packet, the independent variable n was assigned a subsequent value from the list [2, 6, 10, 14, 18]. Additionally, in the session for the ODTS simulator (the 4th example), the number of blocks was fixed to 1 whereas the number of time slots (ts) was a dependent variable calculated as ts = n – 1. Finally, for each packet, the mean execution time and the relative standard deviation were calculated.

Graphs that were acquired in the above-mentioned evaluation followed process showed a linear trajectory for all examples with decentralised data averaging having a somewhat longer startup time. Results also showed relative standard deviation decreased as the number of nodes increased. The graphs showing the mean execution time versus the number of nodes that were gained in this process are presented below in Figures 6, 7, 8 and 9. (To save space, graphs for the relative standard deviation are not shown here.)

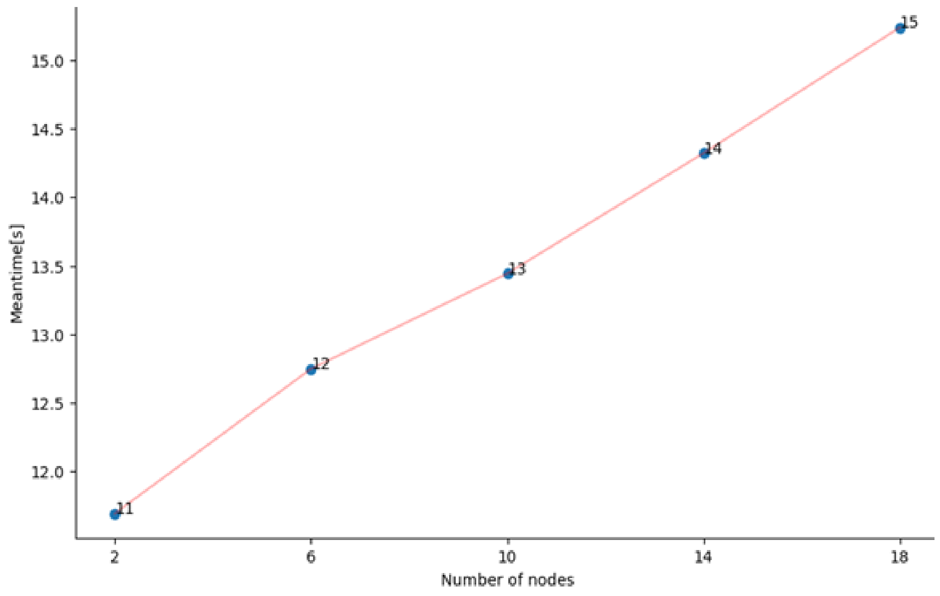


Figure 6*: The federated map's mean execution time.*

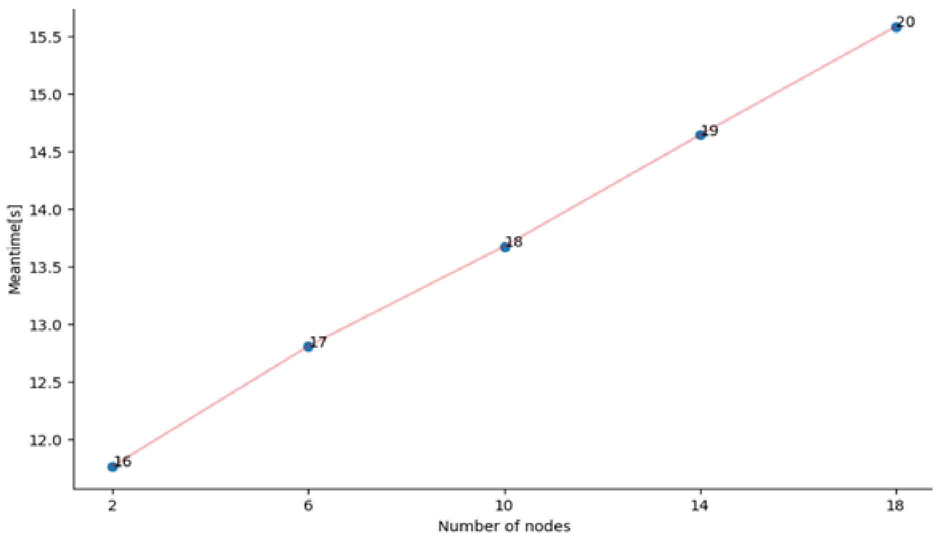


Figure 7*: The centralised data averaging mean execution time.*

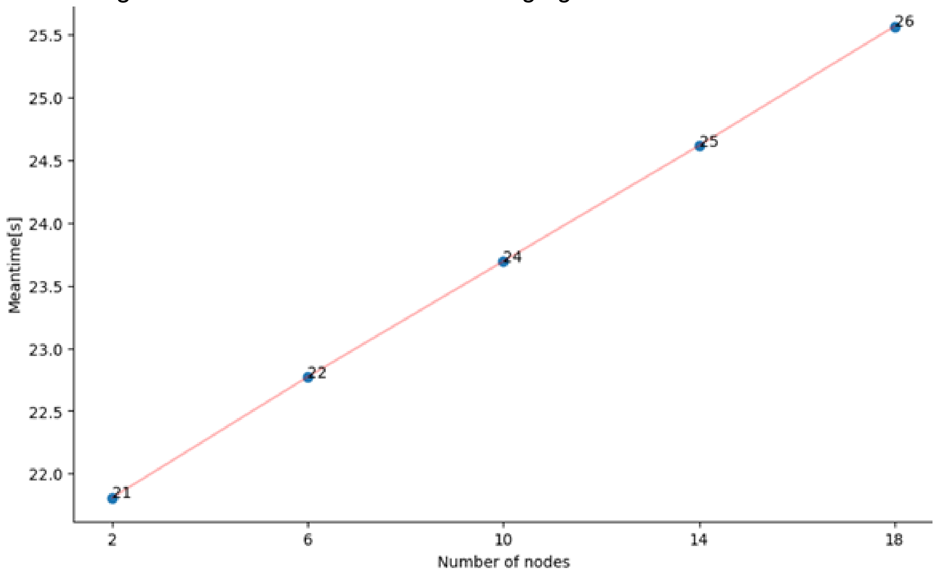


Figure 8*: The decentralised data averaging mean execution time.*

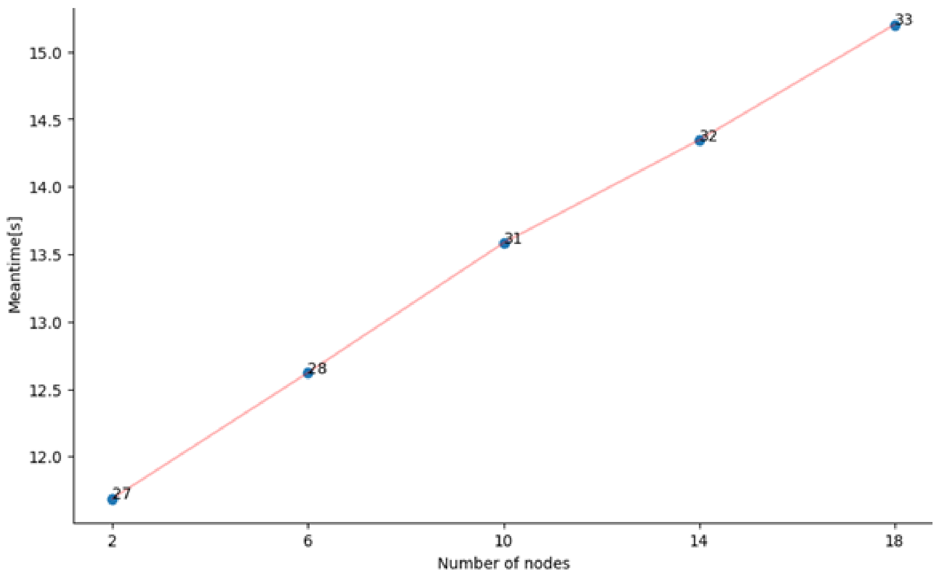


Figure 9*: The ODTS mean execution time.*

2.3.5 Distributed Applications for MNIST NN Training and Testing/Inference on PTB-FLA and MPT-FLA

This section briefly presents the development of two distributed applications for Modified National Institute of Standards and Technology (MNIST) Neural Network (NN) training and testing/inference on PTB-FLA and MPT-FLA, respectively. Here the inference was conducted during the testing of the trained model. (However, generally, both PTB-FLA and MPT-FLA support developing distributed/decentralised swarm applications for both training and inference).

The first application was developed using the four-phase PTB-FLA development parading for humans [26]. In the first phase, the referent sequential code was created by adapting the input code authored by Soham Parmer, a computer engineering student at the University of Waterloo, which is publicly available [36].

The standard MNIST dataset comprises 28x28 pixel digit images, where pixel values are in range 0-255. The referent sequential code uses a copy of the MNIST dataset (publicly available [37]) that comprises 5000 samples, where the last 4000 samples are used for training the MNIST NN and the first 1000 samples are used for testing the trained model. As part of data preparation, the pixel values are transcoded to values in the range 0-1 (by dividing the original values with 255).

For the standard MNIST dataset, the input NN layer has 784 neurons, which corresponds to the digit image size of 28x28 (28x28 = 784), whereas the output NN layer has 10 neurons, which corresponds to the number of digits (0-9). The hidden layer in the referent sequential code has 128 neurons. This NN is traditionally trained by executing a loop wherein the forward propagation, backward propagation, and NN parameters update (using the gradient descent approach) is performed a given number of times (here 100 times). Finally, the trained NN is traditionally tested (using the first 1000 samples) and evaluated by determining its accuracy, which is defined as a ratio of the number of correct predictions/inferences versus the test sample size (here 1000).

In the second, third, and fourth phase, the federated code, the federated code with callbacks, and the PTB-FLA code were developed. As expected, the NN produced in the second phase was slightly different from the NN produced in the first phase, but since this difference was negligible the former became the reference for the rest of the phases. Again, as expected, the third and the fourth phase indeed produced the same NNs as the second phase. The complete PTB-FLA code was successfully performed on the single PC (localhost).

The second application was developed by adapting the first application to (1) the MicroPython (a trimmed down Python version running on small microcontrollers) and (2) the RPi Pico W board (a faithful representative of a platform for smart sensors and Internet of Things - IoTs). To adapt to MicroPython, the code was adapted to use the library uLab library instead of NumPy, because the latter is not supported on MicroPython and the former is a replacement for it.

To adapt to RPi Pico W board memory resources, the size of the input images had to be reduced from 28x28 to 9x9. This optimization was a key to making the second application fit the RPi Pico W board memory footprint. Additionally, because it was not possible to conduct NN training with the large dataset as for the first application (which was done on a PC), the application was reorganised such as to first perform the initial training on the server (running on a PC) with the dataset of 1000 samples, and then to perform the incremental training on clients (one running also on the PC and the other running on the RPi Pico W board) with datasets of only 50 samples, which proved to be sufficient for faster and efficient training feasible for a small platform like the Pico W board.

This approach, besides enabling efficient usage of limited Pico W board resources, as expected, also yielded the final aggregated NN whose accuracy was greater than the accuracy of the NN after the initial training on the server.

The evaluation of the second application was conducted on the configuration comprising one PC (hosting the server and the first client) and one Pico W board (hosting the second client) which were connected over a WiFi router (or a mobile phone acting as an access point). The goal was to evaluate how the accuracy of the final NN depends on: (1) the server's dataset size used in the initial NN training, and (2) the server's number of iterations used in the initial NN training. The dataset size and the number of iterations on clients was fixed to 50 and 300, respectively.

When the server's number of iterations is fixed to 1000, as expected: (1) both the initial and the final NN accuracies steadily increase with the server's dataset size (which is used in the initial NN training), and (2) the final accuracy in better that the initial accuracy, see Figure 10.

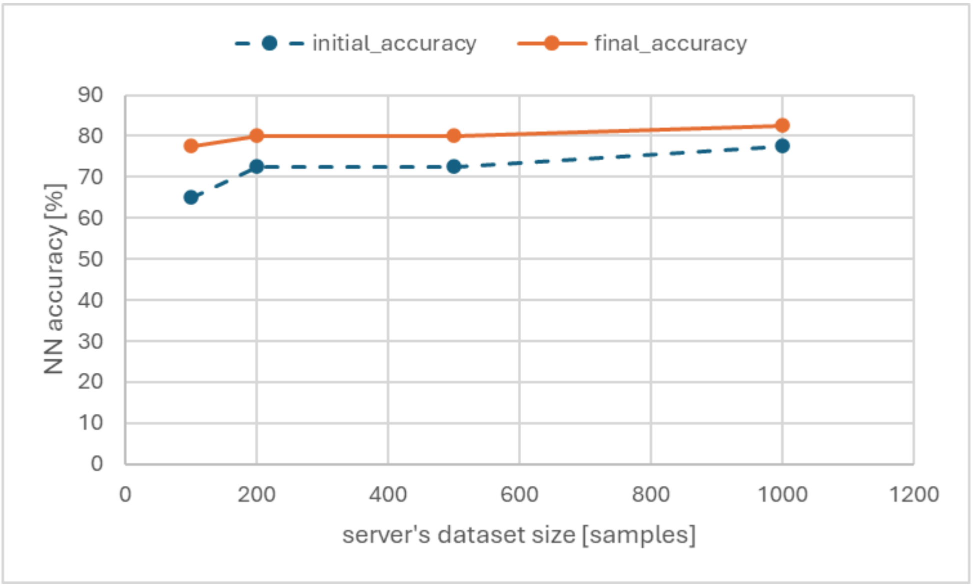


Figure 10*: The initial and the final NN accuracy versus the server's dataset size.*

When the server's dataset size is fixed to 1000, as expected: (1) both the initial and the final NN accuracies rapidly increases with the server's number of iterations (which is used in the initial NN training), and (2) the final accuracy in better that the initial accuracy, see Figure 11.

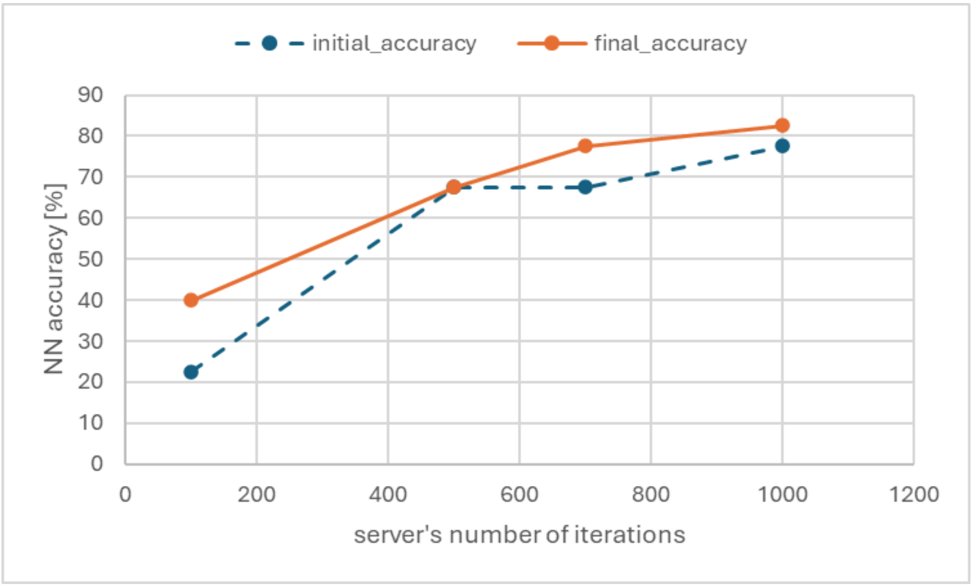


Figure 11*: The initial and the final NN accuracy versus the server's number of iterations.*

These two distributed applications were developed by Marko Nikolovski in his B.Sc. thesis (available in Serbian [38]), mentored by Miroslav Popovic, and are publicly available in the "ptbfla" GitHub repository [35], see modules in the folder example2.

2.3.6 Towards Formal Verification of Federated Learning Orchestration Protocols on Satellites

In this section, we provide a short digest of the research that is to be published by Institute of Electrical and Electronics Engineers (IEEE) Xplore in the Telfor 2024 proceedings, which is currently available as a preprint [31], which comprises the problem formulation and its solution, research achievements, and a highlight of the stochastic Timed Automata (TA) model and formal verification results.

Problem Formulation and Its Solution: The PTB-FLA FL orchestration protocols were formally verified by using the process algebra CSP and the model checker Process Analysis Toolkit (PAT) [27]. The main limitation of [27] is that it is suitable for systems with stationary nodes, but cannot be applied to systems with moving nodes, such as constellations of spacecrafts, where physical timing needs to be considered, which is exactly the main motivation for this paper.

To overcome this limitation, authors of [31] use celestial mechanics to model spacecraft movement, and TA and accompanying tool UPPAAL to formalise and verify the Centralised FL (CFL) orchestration protocol, in two phases. In the first phase, they created a conventional TA model to prove traditional properties, namely deadlock freeness and termination. In the second phase, they created a stochastic TA model to prove the timing correctness (the alignment of spacecraft movement and communication) and to estimate termination probability.

Research Achievements are: (1) the model of a spacecraft movement in the form of the differential equation for the spacecraft's true anomaly that directly follows from Kepler's laws, (2) the conventional TA model of the CFL orchestration protocol, and (3) the stochastic TA model of the CFL orchestration protocol. To the best of the author's knowledge, this is the first paper that uses stochastic TA for formal verification of CFL orchestration protocols on constellations of spacecraft.

Stochastic TA model and formal verification results: The stochastic TA model comprises the stochastic TA model of spacecraft (not shown here for brevity) and the stochastic TA model of the CFL orchestrion protocol in Figure 12.

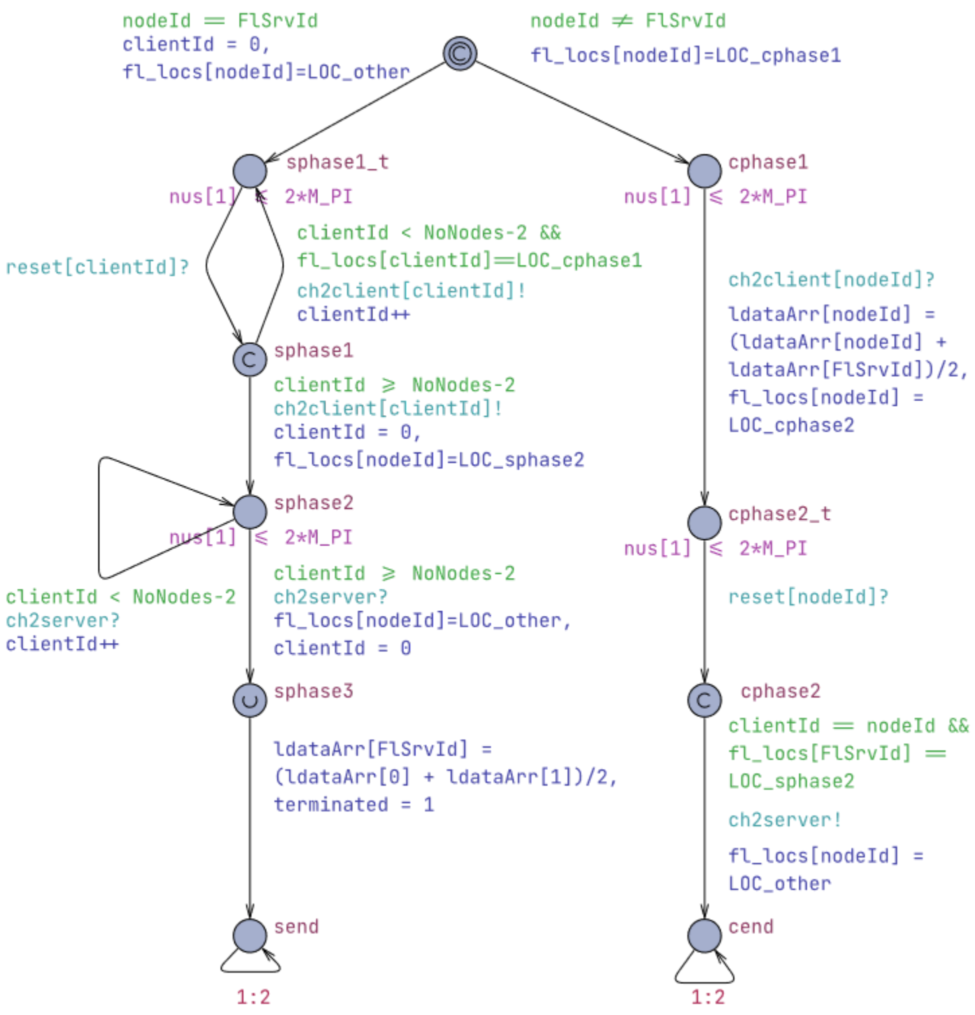


Figure 12*: The stochastic TA model of the CFL orchestration protocol (taken from [31]).*

The stochastic TA model has two main assumptions. Assumption 1: the CFL server instance resides in the Ground Station (GS), whereas fTA the two CFL client instances reside in two spacecrafts (or space vehicles, SVs) which fly on the same orbit with the initial true anomaly *ν* of π and 0, respectively. Assumption 2: each spacecraft should communicate with the ground station when it reaches the periapsis of its orbit.

To check the alignment of spacecrafts' movement and their communication with the ground station we created a simulation query that traces both true anomalies and current locations of the CFL server and the CFL clients. The simulation diagram plot by UPPAAL looks like a Gantt chart drawn atop a timing of analogue signals. The former comprises the three staircase-like lines showing location changes, see Figure 13.

Obviously, the alignment is perfect, see how: (1) the first client (green line) reaches the location *cphase2_t* at the time point *t* = T/2, and the location *cend* at the time point *t* = 3T/2, (2) the second client (blue line) reaches the location *cphase2_t* at the time point *t* = T, and the location *cend* at the time point *t* = 2T, and (3) the server (pink line) reaches the location *sphase2* at the time point *t* = T, and the location *send* at the time point *t* = 2T. Both *nus*[0] and *nus*[1] have the period T, and they start from π and 0 (see the ordinate at the time point t = 0), respectively.

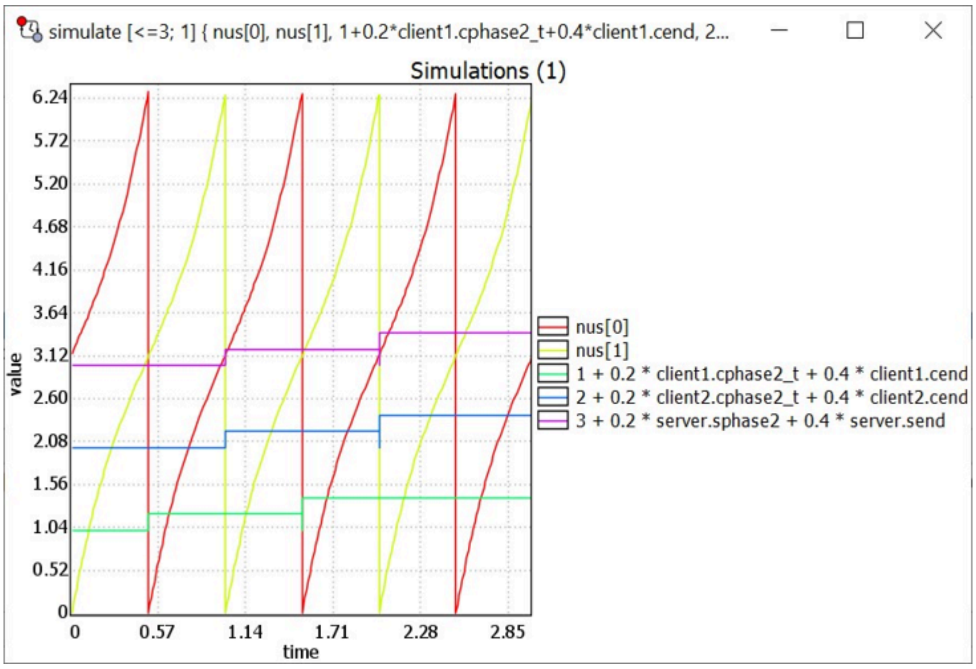


Figure 13*: The UPPAAL tool simulation diagram (taken from [31]).*

For more details on this research see [31].

2.4 Fedra: Advancing decentralised federated learning

Fedra, identified as tool T-WP5-04 in D2.3 [4], is an FL framework that has been developed in TaRDIS in order to support FL in completely decentralised, peer-to-peer swarm systems. In this context, Fedra can be used to train diverse ML models using the local data of each node, while enabling the direct exchange of model parameters amongst the participating peers.

2.4.1 Introduction and Conceptual Framework

FL has emerged as a revolutionary paradigm in machine learning, addressing the growing concerns of data privacy and the challenges of distributed datasets. Traditional FL approaches, while groundbreaking, often rely on centralised aggregators, creating potential bottlenecks and single points of failure. Fedra represents the next evolution in this field, pushing the boundaries of decentralisation, privacy, and efficiency. The general architectural considerations of decentralised federated learning are shown in Figure 14.

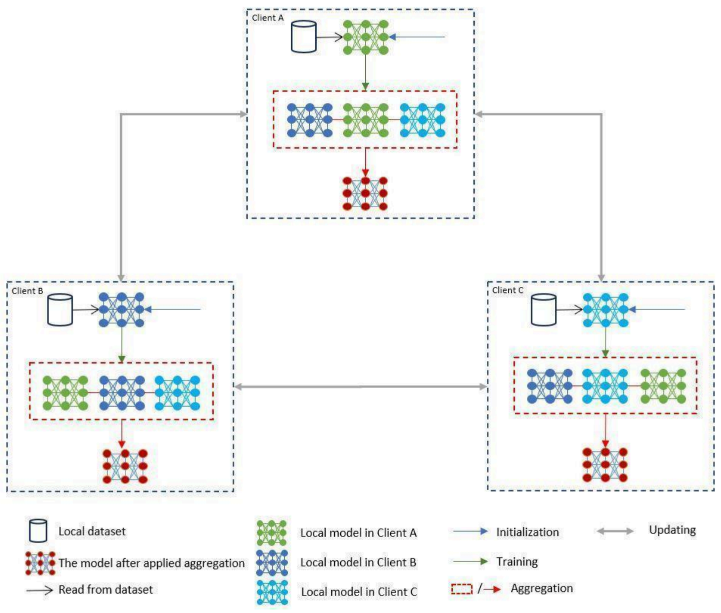


Figure 14*: The general architectural considerations of decentralised federated learning: each node/client trains the model based on the local data and aggregates with the weights of other clients into a global model.*

Fedra is not just another FL framework; it's a reimagining of how collaborative machine learning can be achieved in a truly decentralised manner. By leveraging peer-to-peer (P2P) communication through libp2p, Fedra creates a robust, scalable network where each node is an equal participant in the learning process.

2.4.1.1 Key Innovations:

- **True Decentralisation**: Unlike systems that claim decentralisation but still rely on central coordinators, Fedra eliminates all central points of control. Each node in the Fedra network is fully autonomous, capable of training, aggregating, and contributing to the global model without centralised oversight.
- **Enhanced Privacy Guarantees**: Fedra takes privacy a step further than traditional federated learning. Not only does raw data never leave local devices, but the peer-to-peer nature of communications means that even model updates are shared in a more private, directed manner.
- **Adaptive Learning Topology**: The network in Fedra isn't static. It can dynamically adjust based on node availability, network conditions, and even the nature of the learning task at hand. This adaptivity ensures resilience and performance across various scenarios.
- **Model Agnosticism Redefined**: While many frameworks claim model agnosticism, Fedra's architecture is designed from the ground up to accommodate not just different model architectures, but entirely different learning paradigms. From simple neural networks to complex Long Short-Term Memory (LSTM) models, Fedra's flexibility is unparalleled.

2.4.2 Conceptual Architecture of Fedra

Fedra's architecture is a carefully orchestrated symphony of components, each playing a crucial role in the decentralised learning process:

1. **P2P Communication Layer (P2PHandler)**:
    - Acts as the nervous system of the Fedra network.
    - Manages all inter-node communications, from initial peer discovery to ongoing model update exchanges.
    - Utilises libp2p to ensure secure, efficient, and anonymous peer-to-peer interactions.
2. **Data Management and Preprocessing (DataLoaderHandler)**:
    - Serves as the sensory input system, preparing and feeding data to the learning models.
    - Handles diverse data types and structures, ensuring compatibility across different learning tasks.
    - Implements advanced preprocessing techniques to optimise learning efficiency.
3. **Core Operations Module (Operations)**:
    - Functions as the brain of each node, performing critical computations.
    - Manages serialisation and deserialization of model updates, crucial for efficient network transmission.
    - Implements the decentralized federated averaging algorithm, the key to collaborative learning in a decentralised setting.
4. **Network State Management (NetworkState)**:
    - Acts as the collective memory of the network.
    - Keeps track of the status and contributions of all participating nodes.
    - Enables informed decision-making for adaptive learning strategies.
5. **Orchestration Engine (Main Script - fedra.py)**:
    - Serves as the conductor, coordinating all components to work in harmony.
    - Manages the lifecycle of the learning process, from initialization to convergence.
    - Implements high-level learning strategies and protocols.

This architectural design ensures that Fedra is not just a tool for federated learning, but a comprehensive ecosystem for decentralised, collaborative AI development.

2.4.3 Low-level Fedra's Architecture and Process - Architectural Components in Detail

2.4.3.1 P2PHandler (handler.py)

The P2PHandler is the cornerstone of Fedra's decentralised nature. It leverages libp2p to create a robust, secure, and efficient peer-to-peer network. The main task of the Fedra tool is shown in Figure 15.

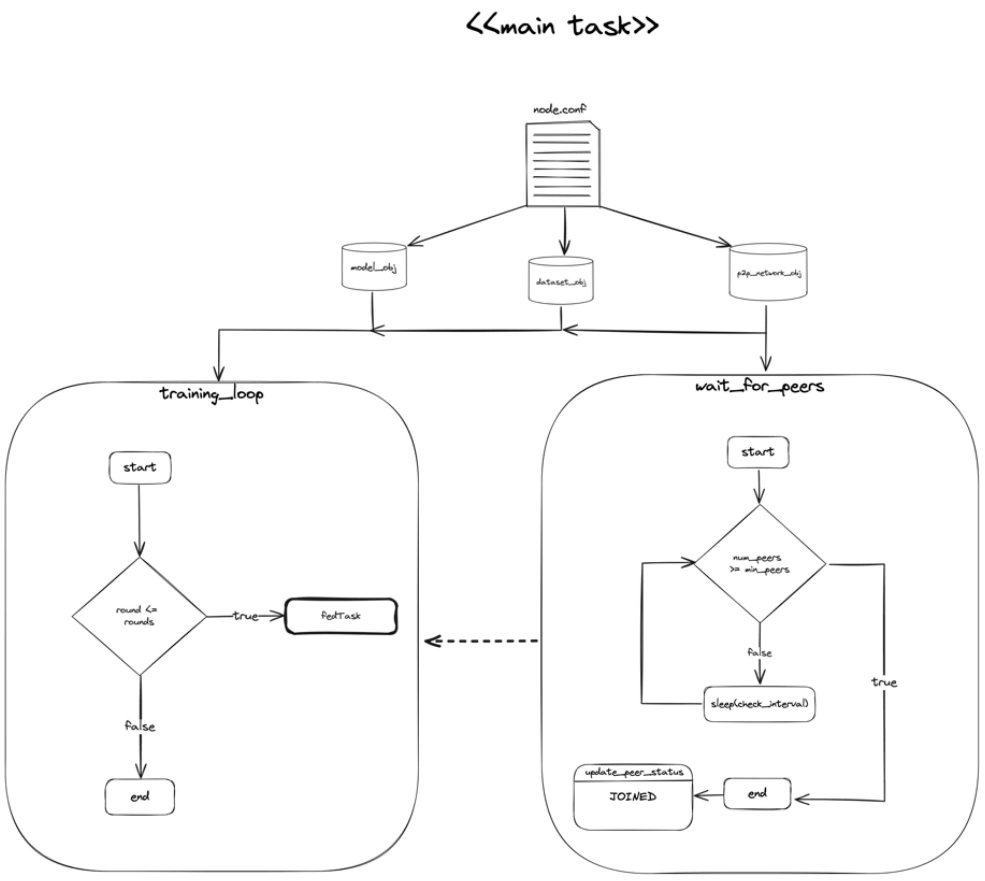


Figure 15*: The main task of Fedra tool, depicting the node.conf file, as well as the wait for peers' workflow (in order to initiate the training process) and the training loop workflow (that performs the federated learning process).*

Key Features:

- **Dynamic Peer Discovery**: Utilises libp2p's peer routing to dynamically discover and connect to other nodes in the network.
- **Publish-Subscribe System**: Implements a sophisticated pub-sub mechanism for efficient broadcast of model updates and network states.
- **Message Chunking**: Handles large model updates by breaking them into manageable chunks, ensuring smooth transmission even in bandwidth-constrained environments.
- **State Synchronisation**: Maintains network-wide consistency through periodic state broadcasts and reconciliation.

2.4.3.2 DataLoaderHandler (process.py)

This component is crucial for Fedra's ability to handle diverse datasets and model types.

Key Features:

- **Adaptive Data Loading**: Supports various data formats and structures, from simple CSV files to complex time-series data.
- **Preprocessing Pipeline**: Implements a flexible preprocessing pipeline that can be customised based on the specific requirements of each learning task.
- **Batching Strategies**: Offers advanced batching techniques to optimise memory usage and training efficiency.
- **Data Privacy Enhancements**: Incorporates privacy-preserving techniques like differential privacy at the data preparation stage.

2.4.3.3 Operations (operations.py)

The Operations class is the computational powerhouse of Fedra, handling critical tasks in the federated learning process.

Key Features:

- **Efficient Serialisation**: Utilises advanced serialisation techniques to minimise the size of transmitted model updates.
- **Federated Averaging**: Implements a robust decentralized federated averaging algorithm [39] that can handle updates from multiple peers, potentially with varying contributions. In the current version of Fedra, we assume that all the clients in the federated framework exhibit peer2peer connections.
- **Weight Compression**: Incorporates weight compression techniques to further reduce communication overhead.
- **Anomaly Detection**: Includes mechanisms to detect and handle anomalous or malicious updates, enhancing the security of the learning process.

2.4.3.4 NetworkState (state.py)

This component maintains a comprehensive view of the network's state, crucial for informed decision-making in a decentralised environment.

Key Features:

- **Peer Status Tracking**: Maintains real-time status of all peers, including their training progress and contribution quality. There are several methods for sharing model weights in Decentralised FL (DFL) based on different protocols, e.g., pointing, gossip, or broadcast protocols. We assume here that each peer broadcasts its state to the rest of the peers through a Publish-Subscribe (Pub/Sub) system.
- **Weight Version Control**: Implements a versioning system for model weights, allowing for rollback and conflict resolution.
- **Performance Metrics**: Tracks and analyses network-wide performance metrics to guide adaptive learning strategies.
- **Fault Tolerance**: Incorporates mechanisms to handle peer dropouts and rejoin scenarios seamlessly.

The two asynchronous tasks that run in the background are shown in Figure 16.

Funded by the European Union

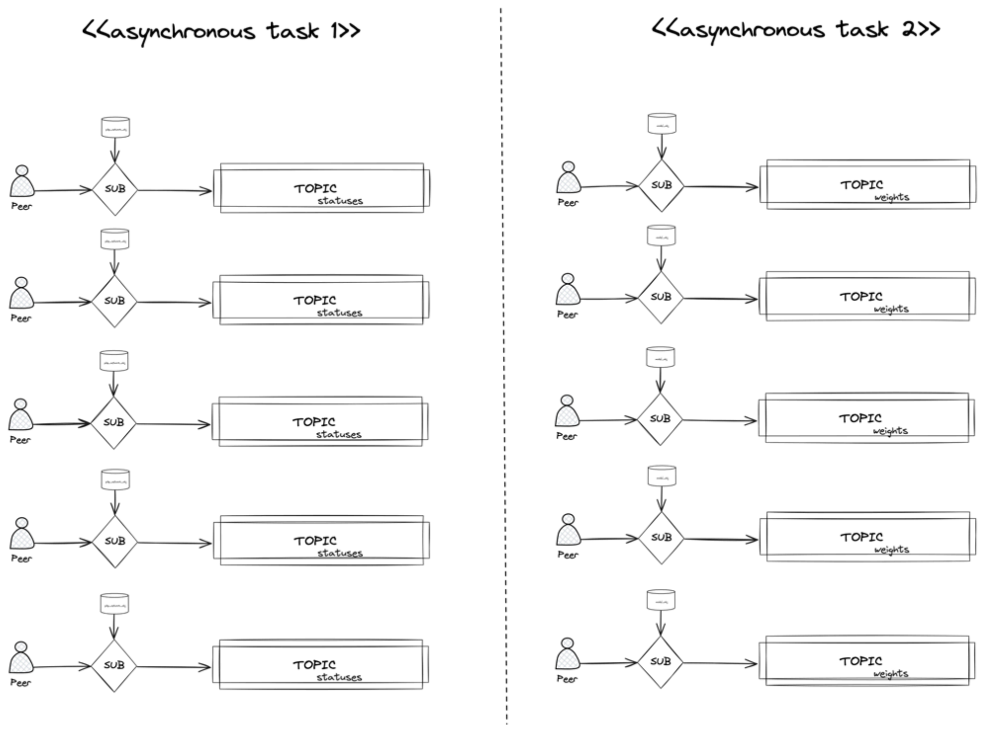


Figure 16*: The two asynchronous tasks that run in the background: the peers publish their status in the respective topic (left) and their model weights (right).*

2.4.4 The Federated Learning Process in Fedra

Fedra's learning process is a sophisticated dance of distributed computation and collaborative model improvement:

1. **Network Initialization**:
    - Nodes join the P2P network using libp2p's peer discovery mechanisms.
    - Each node broadcasts its initial status and capabilities to the network.
    - The network collectively establishes initial parameters like learning rate and batch size based on the capabilities of participating nodes.
2. **Data Preparation and Local Training**:
    - Nodes use the DataLoaderHandler to preprocess their local datasets.
    - Initial model architectures are either predefined or negotiated based on the collective dataset characteristics.
    - Each node performs local training, with the flexibility to use custom optimizers and loss functions suited to their data.
3. **Model Update Exchange**:
    - Post-training, nodes serialise their model updates using the Operations class.
    - Updates are strategically disseminated through the network using a combination of direct peer connections and gossip protocols.
    - The P2PHandler manages the chunking and reassembly of large model updates to ensure reliable transmission.
4. **Decentralised Aggregation**:
    - Nodes perform local aggregation of received model updates using the federated averaging algorithm in the Operations class.
    - The aggregation process is weighted based on factors like peer reputation and data quality, as tracked by the NetworkState.

- Anomaly detection mechanisms filter out potentially harmful or low-quality updates.

5. **Adaptive Learning and Convergence**:
   - The process iterates, with each round potentially adapting parameters based on network performance.
   - Convergence is determined through a decentralised consensus mechanism, considering factors like model performance, update magnitude, and network stability.
   - The NetworkState component plays a crucial role in coordinating this decentralised decision-making process.
6. **Continuous Evaluation and Refinement**:
   - Throughout the process, nodes continuously evaluate the global model on their local validation sets.
   - Feedback loops allow for dynamic adjustment of learning rates, batch sizes, and even model architectures.
   - The network can seamlessly handle the joining of new peers or the departure of existing ones, ensuring robustness and scalability.

The federated learning workflow and the shared objects among the peers participating in the FL process are shown in Figure 17 and 18.

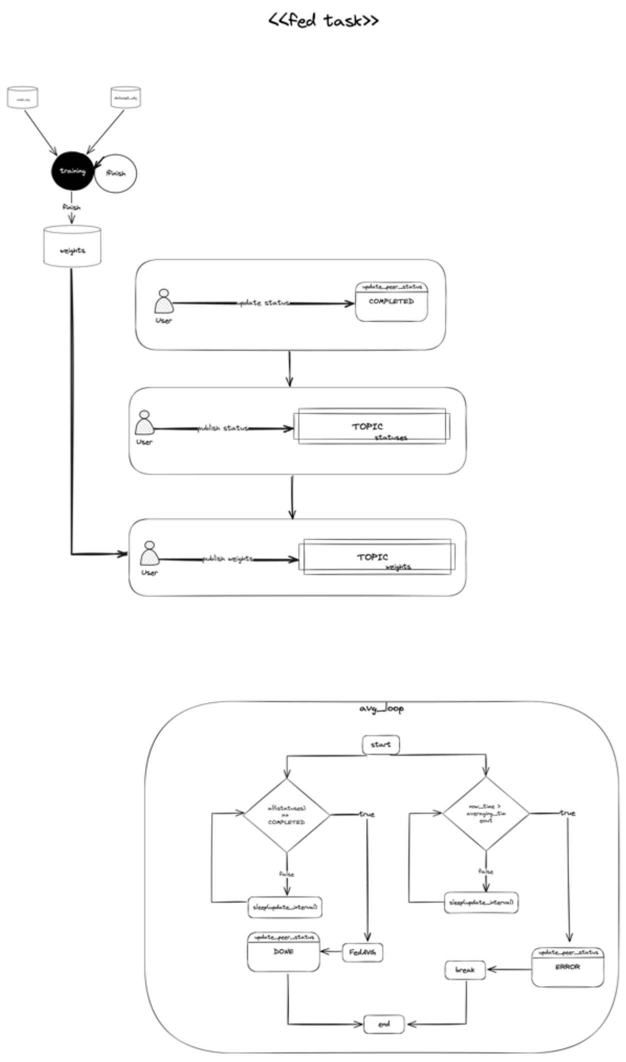


Figure 17*: The federated learning workflow,*

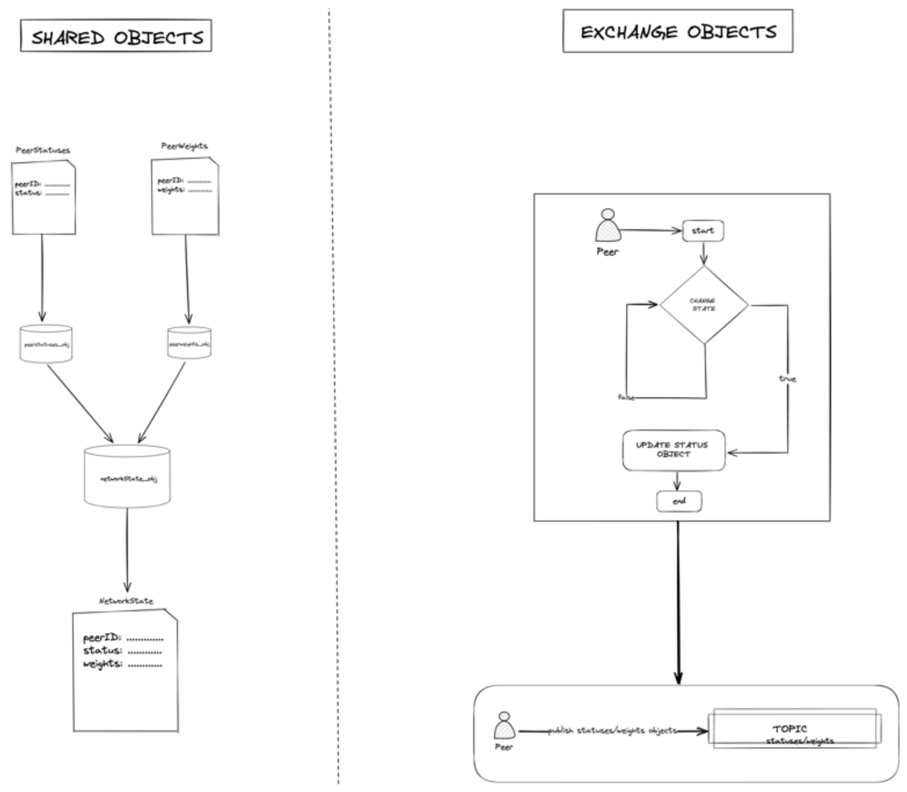


Figure 18*: The shared objects (status and weights) among the peers participating in the FL process.*

2.4.5 Advanced Features and Future Directions

Fedra's architecture is designed with extensibility in mind, paving the way for advanced features and future enhancements:

- **Differential Privacy Integration**: Plans to incorporate more advanced differential privacy techniques at both the data and model levels.
- **Heterogeneous Hardware Support**: Developing capabilities to optimally utilise diverse hardware configurations across the network, from edge devices to high-performance clusters.
- **Multi-Task Learning**: Extending the framework to support simultaneous training of multiple, possibly related models across the network.
- **Blockchain Integration**: Exploring the integration of blockchain technology for immutable record-keeping of model updates and peer contributions.
- **AI-Driven Network Optimization**: Implementing AI algorithms to dynamically optimise the network topology and learning parameters for maximum efficiency.

Fedra represents not just a step, but a leap forward in the field of federated learning. By reimagining the very architecture of collaborative learning, Fedra opens up new possibilities for privacy-preserving, efficient, and truly decentralised AI development. As the framework continues to evolve, it promises to push the boundaries of what's possible in distributed machine learning, paving the way for a new era of collaborative AI that respects privacy, embraces diversity, and harnesses the true power of decentralised computation.

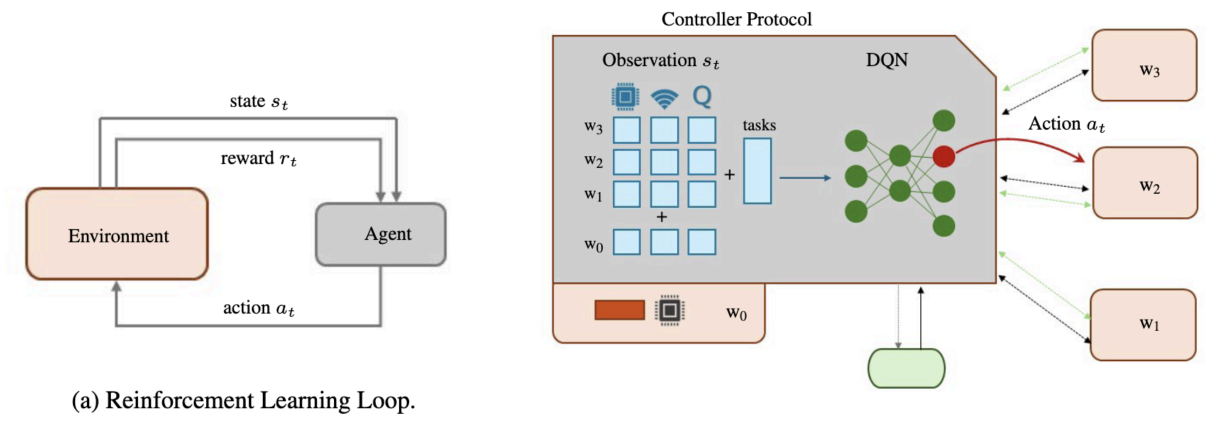
3 Advances on AI-driven planning, deployment and orchestration framework

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

PeersimGym is a simulation environment designed to address the task offloading problem using Reinforcement Learning (RL). Task offloading refers to the process of transferring computationally intensive tasks from resource-constrained devices to less-strained devices. As described in [40], it allows for the evaluation and optimization of RL algorithms under dynamic network conditions, facilitating efficient task distribution and resource utilisation. The framework integrates various RL strategies to solve complex offloading scenarios, promoting better performance in network management and operations. [40].

The benefit of RL is that it enables learning optimal offloading strategies through iterative interactions. However, it requires access to rich datasets and custom-tailored, realistic training environments. Figure 19 shows the general overview as follows: The interaction cycle of an RL agent with its environment is structured around a continuous loop where, at each timestep t = 1, ..., T, the agent observes the system state st, executes an action at based on this observation, and receives feedback in the form of a reward rt. This feedback reflects the effectiveness of the action, taking into account both its immediate impact and its influence on future states. Through this iterative process, the agent refines its policy - a set of rules determining its actions in various states - to maximize cumulative rewards, thereby aligning with the goal of optimizing task offloading decisions. In Figure 19.a we have the common RL interaction cycle, obeyed by the controller agents. The cycle begins with the controller observing a state, as represented in Figure 19.b, containing information about the part of the environment observed by the controller including its own properties and gathered information on its neighbourhood. The controllers then make an offloading decision, the actions taken and the inherent dynamics of the environment produce a state transition and the cycle repeats itself. In the multi-agent setting all the agent's would have a similar interaction loop at the same time, resulting in a joint-action.

PeersimGym is an open-source, customizable simulation environment, with the purpose of developing and optimising task offloading strategies. It supports a wide range of network topologies and various constraints. It also integrates a PettingZoo-based interface for RL agent deployment in solo and multi-agent setups. It has been extensively discussed in D5.1 and presented in the ECML paper [40]. In the last period, there were several code fixes motivated by tool usage outside of TaRDIS.



(b) Local state and action for a worker node.

Figure 19*: General and problem-specific RL state action overview.*

3.2 FAuNO: Federated AI network orchestrator

FAuNO, identified as tool T-WP5-05 in D2.3 [4], represents a federated AI network orchestrator that enhances the coordination and operation of distributed AI systems. Leveraging FL principles, FAuNO ensures privacy-preserving collaboration across multiple nodes while maintaining high performance and scalability. This orchestrator is designed to manage AI workloads efficiently, balancing computational demands and ensuring optimal resource use across the network.

The objective of this tool is to learn to orchestrate the network in a federated way. It is designed to optimise computer network orchestration through Federated Reinforcement Learning (FRL). The main goals are to enhance efficiency and performance of distributed network systems and to provide a robust and scalable solution for managing and optimising network operations across decentralised nodes. It addresses challenges as latency, bandwidth constraints and data privacy concerns. Therefore, the main benefits can be identified as the following:

- The nodes learn and adapt locally while contributing to a global model
- Data privacy is ensured.
- The need for extensive data transfer is reduced.
- The latency and bandwidth usages are minimised.

The FAuNO tool interactions with other network entities can be displayed as in Figure 20 (for more details, see D2.3).

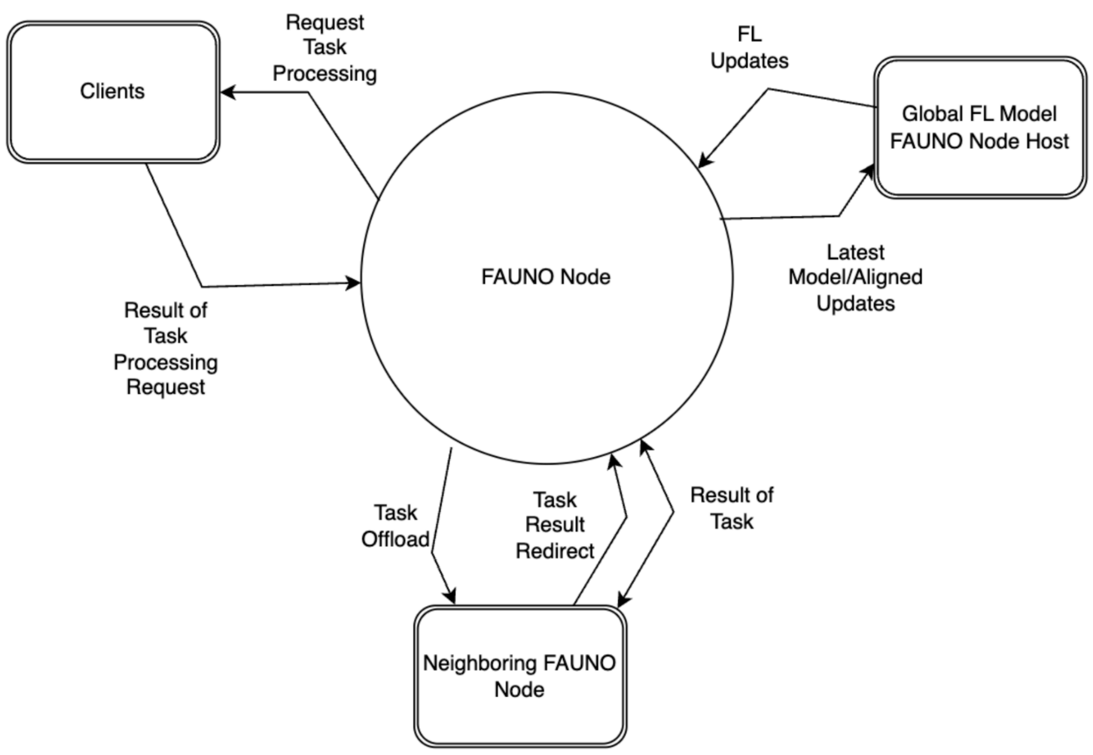


Figure 20*: FAuNO tool interaction diagram.*

3.2.1 FAuNO Overview

FAuNO as a Federated solution utilises a Client/Server architecture to have the participants in the federation cooperate in training a global model utilising their local individually learned experiences. The global model is handled by a FAuNO node that is capable of hosting a component to manage the receival of updates and their aggregation into the global model, we call this as the Global Model Host FAuNO Node, or just global model host. The global model host other than the module that manages the global model is just like any other FAuNO node, having its own separate local model to learn locally how to best orchestrate the node's workload. The functions of a worker node are to process the tasks that arrive on the node, share their own state information with their immediate neighbours, collect information about their neighbourhood and make task offloading decisions based on the collected information. In Figure 21 we can observe a diagram with the components that make up each of the FAuNO node types, which include four components. Three of which are hosted by all the nodes. These are the data processing layer responsible for execution of the tasks assigned to the node. The data collection layer, responsible for communicating with the node's neighbours and managing the collected information. And the FRL Client that is responsible for the independent orchestration of the node with a task offloading mechanism. The last component is hosted by a single agent in the network and is the FRL Server that manages the federation of FRL Clients, when training a model that can solve a global objective considering the experiences of all the members of the federation.

We focus on the orchestration mechanism leaving the data collection and task processing generic to accommodate varied scenarios.

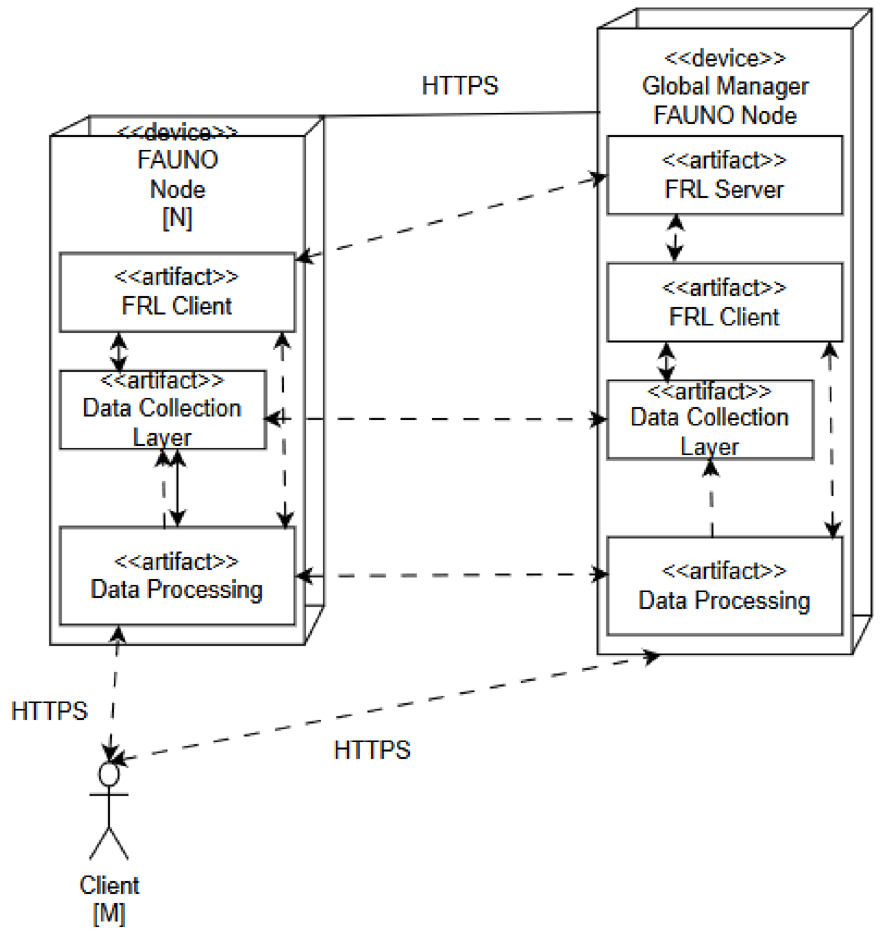


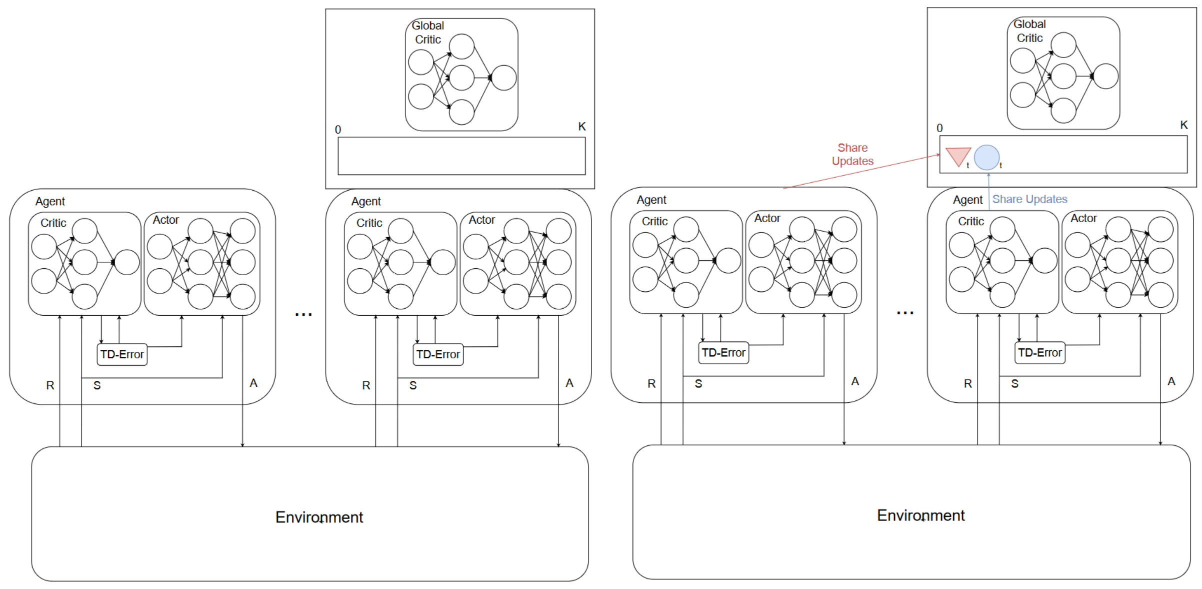
Figure 21*: FAuNO Node vs FAuNO Global Manager.*

3.2.2 FAuNO orchestration mechanism

Following common client\server federated algorithms we divide the FAuNO orchestration into two components: the global training and the local training. We utilise Proximal Policy Optimization (PPO) [41] for the local training component. PPO is an algorithm from the Actor-Critic family of RL algorithms where two components exist, an actor that learns a mapping from observations of the state directly to a distribution over the possible actions that can be taken. And, a Critic component, that is responsible for evaluating the Actor's choices. In particular the PPO algorithm looks to improve the reliability and stability of the training by limiting the size of the updates that can be taken during learning. We utilise function approximators for both the actor and the critic.

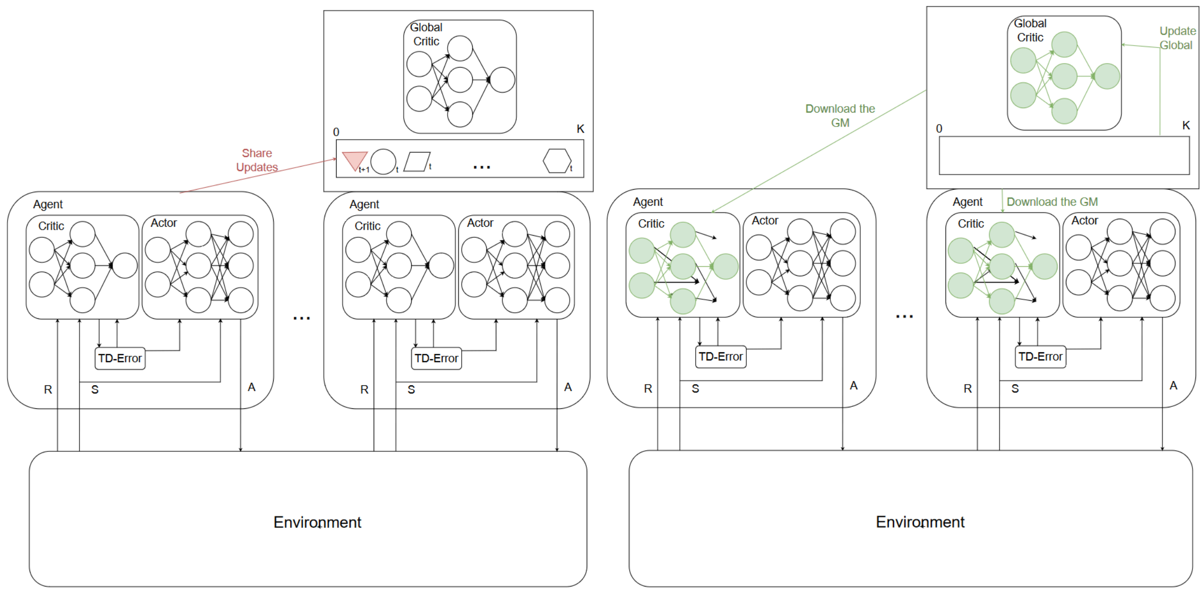
For the global training, where we wish to utilise the federated agent's learned knowledge to cooperate in training a global model. To do so we simplified and built upon the FedBuff solution [42], as demonstrated in Figure 22.

Funded by the European Union



a) Beginning of a round. All nodes download the same model

b) First updates arrive.



c) When multiple updates of one agent arrive, the old one is swapped.

d) When k updates from distinct agents arrive, the Global model is updated.

Figure 22: *Modified FedBuff solution.*

The federated buffering solution used is a buffered asynchronous method where the global server waits for the first K-updates instead of all the participants before the collected local updates are aggregated and used to update the global model. In our setting, we extend the mechanism to accommodate continuous control and have the agents continue training, and sending updates to the global model, these updates replace the last update sent and are weighted more when aggregating. When enough updates are received, we consider a FedAvg [43] like aggregation of the updates to update the global model. All the participants then download the global model. This process is illustrated in Figure 22.

3.2.3 Preliminary Results

We have tested our solution on the PeersimGym environment pursuing two preliminary research questions while doing Binary Task offloading.

3.2.3.1 Baselines

We compare the FAuNO orchestration mechanism to a set of baselines including heuristic algorithms:

- Least Queues: Tasks are always sent to the nodes with the smaller percentual queue
- Random: Tasks are Randomly offloaded
- No offloading: (Always local in plots) Tasks are only processed locally

And a set of Reinforcement Learning algorithms used in a Multi-Agent setting:

- Double Deep Q-Network [44]
- Advantage Actor Critic [45]
- Proximal Policy Optimization [41]

3.2.3.2 Metrics

We consider three commonly used metrics. The number of times a node exhausted their resources and was left without room for more tasks, we call this state being overloaded. The number of tasks dropped. And the average time needed from task request until results receival.

3.2.3.3 Testing scenarios

We observe these metrics on two testing scenarios, we consider a scenario with an increasing number of clusters of nodes as generated by the ether tool integrated with PeersimGym[46], and a fixed task arrival rate at each node. We also compare our algorithm on a scenario with a fixed topology and an increasing task arrival rate. The topologies with two clusters can be observed in Figure 23. An in-depth explanation of the PeersimGym visualization tool used to obtain Figure 23 is available on the PeersimGym's repository.

Funded by the European Union

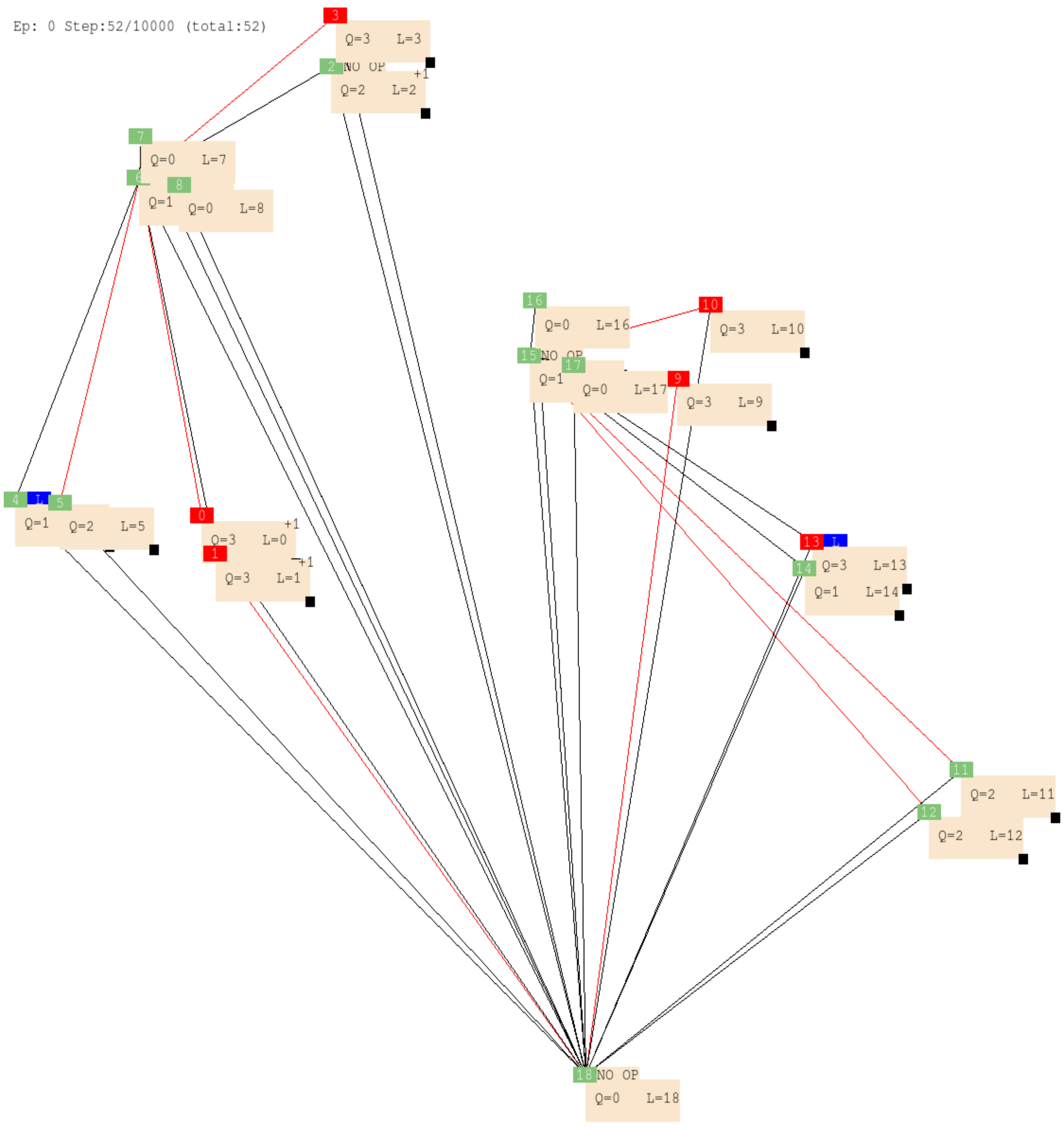


Figure 23*: Topology with 2 clusters of nodes as generated by Ether.*

As can be seen in Figure 24 andFigure 25, the FAuNO orchestration matched the baselines in the scenarios considered on both number of overloads and dropped tasks and outperformed the baselines when considering the response time.

Funded by the European Union

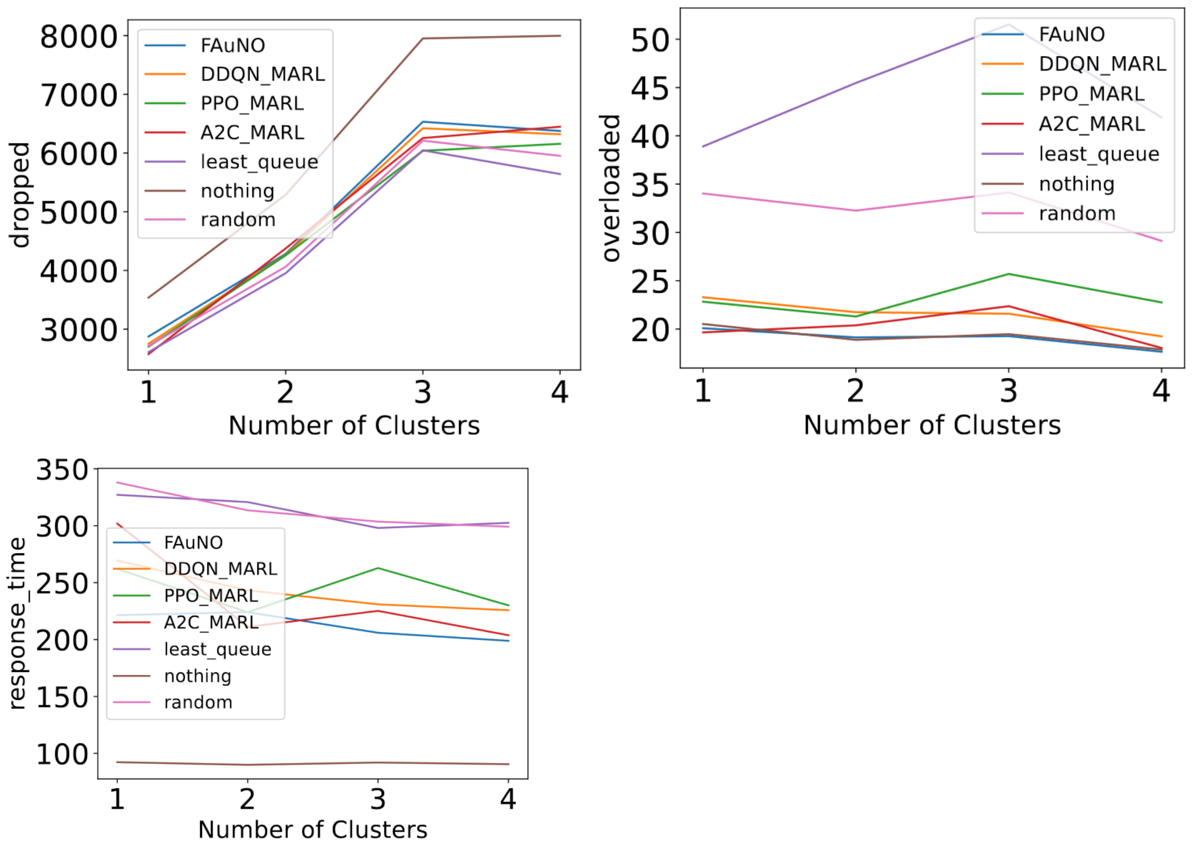


Figure 24: *Results of Scenario with increasing number of clusters.*

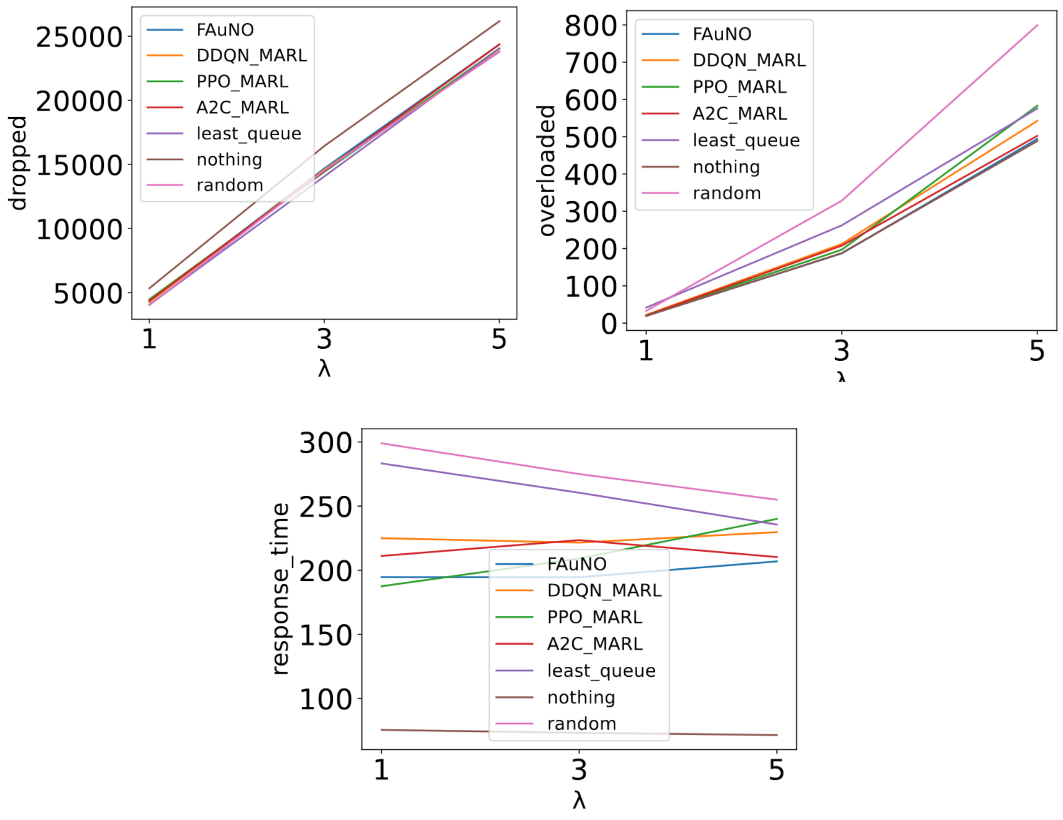


Figure 25*: Results of Scenario with increasing task arrival rates.*

4 Advances on lightweight, energy-efficient ML techniques

In the previous deliverables, we had provided a theoretical overview of the main neural network compression techniques and reviewed some of the literature. In the meantime, we developed some initial code for applying those techniques in real problems. Therefore, during the second year of the activity, the focus was on the implementation of the ML lightweight tools that will be integrated in the TaRDIS toolbox. The lightweight ML tools were identified in the TaRDIS architecture in D2.3 [4] as T-WP5-06 (early exit), T-WP5-07 (knowledge distillation) and T-WP5-08 (pruning). We also published some results regarding theoretical and practical advances that can be of relevance regarding lightweight, energy-efficient ML techniques. We considered an Over-the-Air Computation scheme for fast data aggregation in [47]. Additionally, we proposed a joint adoption of FL principles and the utilization of the Over-the-Air computation wireless transmission framework [48]. We also published lightweight inference by neural network pruning [49], where the application of neural networks in resource constrained edge-devices was addressed. We proposed an IoT-Edge-Cloud computing system, designed for multiple smart homes in [50], and a two-fold FL scheme for improving privacy, EE and communication-efficiency of future 6G maritime networks in [51].

4.1 Pruning

The development of the Pruning method (T-WP5-08) and its application on sample datasets was conducted during this period. In the initial stage of the TaRDIS project, we had developed a wrapper function for applying pruning on PyTorch models, using the Neural Network intelligence (NNI) Microsoft library. While this yielded results, there were some disadvantages, such as the fact that this library does not support pruning of LSTM models.

For that reason, we also explored the usage of TensorFlow lite, a very popular library for lightweight neural networks, and applied pruning to an LSTM model for energy consumption, for a variety of pruning rates. The results were satisfactory, as there was a reduction in file size and inference time, with a minimal effect on the Mean Squared Error (MSE). In some cases, such as the 40 % pruning rate, we see an improvement, whereas the memory required for inference can be reduced without increasing significantly the mean squared error. The results of the pruning process are shown in Figure 26 and 27.

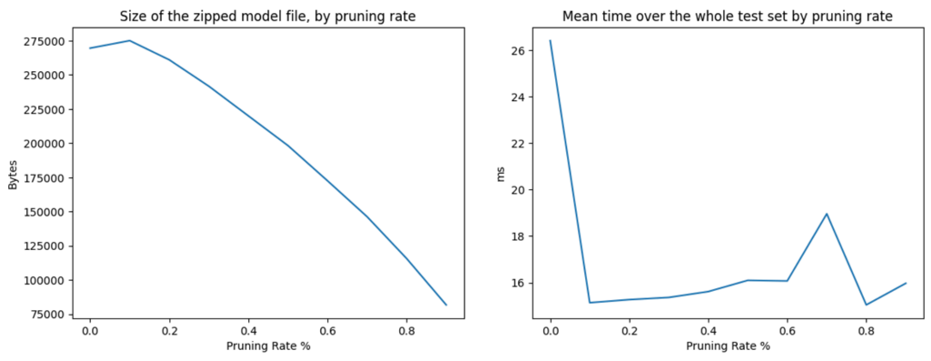


Figure 26*: Results of the pruning process: size of the zipped model (left), mean inference time (right)*

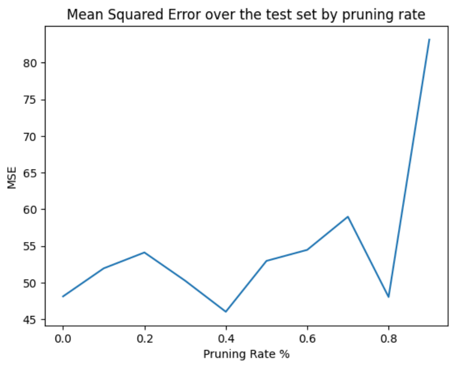


Figure 27*: Results of the pruning process: mean squared error wrt the pruning rate.*

4.2 Early exit of inference

During the second year of the TaRDIS activity, the development of the early-exit technique was conducted in a tool that will be included in the TaRDIS toolbox (T-WP5-06) and its implementation on sample datasets. Furthermore, a tool for the deployment of the several parts of the early exit models was designed and developed, named decentralized exit or D-exit, as reported in the following section.

Developing an early exit framework was more challenging, as there are no libraries that can be used out of the box to split a network into sub models. The first thing the user has to do is separate the model into sub-modules that are contained in a nn.Sequential container. The created class infers the output shape at the end of each Module and creates an Artificial Neural Network that serves as the exit at each.

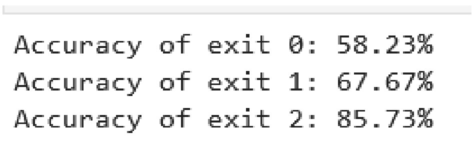
Three Functions to train the network have been created:

One that trains the network as a whole (all the exits together). It has been noticed that the exits closer to the first layers get trained more, when the whole network is trained together. This can be due to some effect of the vanishing gradient problem. As the first exit is close to the input layers, the loss function gets multiplied less times than the ones further away, leading to a smaller gradient and thus a smaller effect on the training of the model. Additionally, the loss function from two exits could either cancel each other out, or make the loss overshoot its actual target.

For those reasons, the two other functions were designed to train a) just the exits and b) only one exit at a time. These functions can be very useful, as they can be applied to a pretrained model, without messing with its training.

We tested the CIFAR-10 [52] dataset with a pretrained visual geometry group (vgg) type model and the best results were observed using the third method of training each exit independently.

Here we can see the accuracy of each exit, if all samples were to take that specific exit.



```
Accuracy of exit 0: 58.23%
Accuracy of exit 1: 67.67%
Accuracy of exit 2: 85.73%
```

Curiously enough, when a combination of the exits was used, the overall accuracy reached 88% in the classification problem. The histogram showing the number of times that the model exits on different exits is presented in Figure 28.

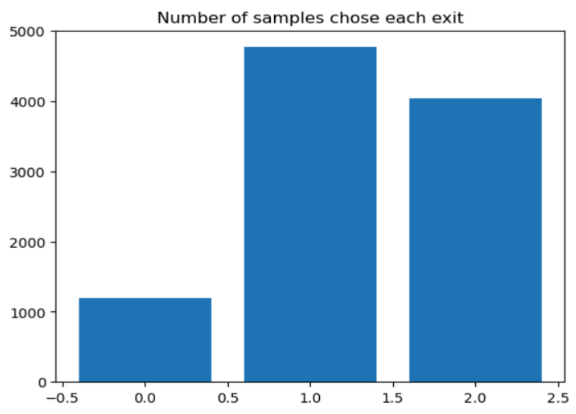


Figure 28*: Histogram depicting the number of times that the model exits on exit 1 (approx 1000), exit 2 (approx 5000) and final exit 3 (approx 4000).*

Finally, a function to split that entire model to smaller ones has been developed, in order for one to be able to deploy each sub model independently. This functionality will be paired with the DEXIT development (see following sections below).

Most of the above functionalities have been abstracted from the user. That being said, one has to train the model themselves, as it is impossible to know beforehand the correct combination of loss function, optimizer learning rate and other hyperparameters, for each and every problem.

This means that, at least for now, a basic understanding of machine learning is required for someone to use the early-exit functionality of the TaRDIS toolkit.

4.3 Knowledge distillation

Similarly to the previous two methods, the development of the knowledge distillation technique (T-WP5-07) was conducted during this reporting period, along with the initial testing on sample datasets. As for now, our knowledge distillation development consists of a function, that imitates a normal PyTorch training loop, but instead of calculation the loss function, between the output of the student model and the actual labels, it is a combination of the labels and the outputs of the teacher model, filtered through a softmax temperature function.

Apart from the usual hyperparameters that one has to select, in Knowledge distillation two extra variables are present: (i) the percentage which the teacher takes into account. While training, a combination of the actual labels and that produced by the teacher can be used; (ii) additionally, the temperature in the softmax temperature function, which determines how sharp the teachers' outputs will be.

As a proof of concept, we trained a small model for CIFAR-10 classification. The first column depicts the model that was trained the conventional way and the other models trained with different temperature values. The teacher that was used was a pretrained model found on github, with 89% accuracy. The results of the knowledge distillation process are displayed in Figure 29.

As we can see the optimal value for this specific configuration is 3, that outperforms the original model, by a small margin. Future modifications to the distillation training loop could be a conditional usage of the teacher, if it correctly classified the sample, so that it does not misguide the student.

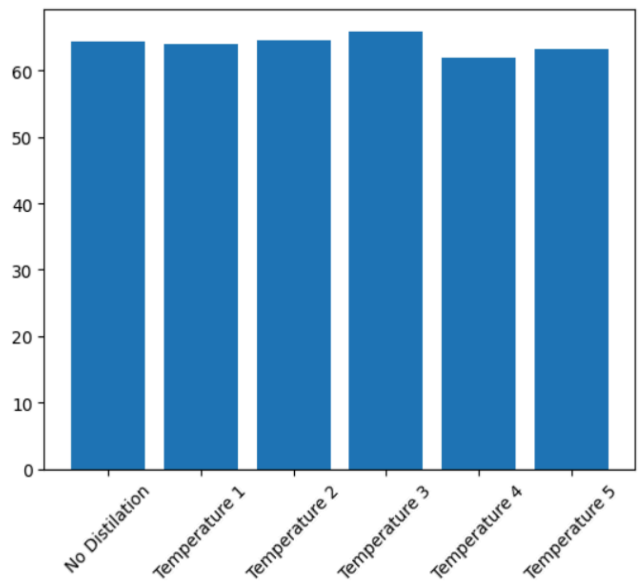


Figure 29*: Results of the knowledge distillation process: accuracy of the student model compared to the original teacher model for different temperatures.*

Regarding the measured efficiency of the Knowledge Distillation (KD) and the EE methods, most papers measure the success of their methods by compression rate, usually expressed as a percentage of weights removed from the original model. But measuring compression this way, leaves us with a big question. What if, instead of training a bigger model and then compressing it, we instead created and trained a small model from the start. That is the comparative analysis that we demonstrate here. The models are first separated in what is called iteration. Each iteration has 10% less weights than the previous one. In Figure 30, we can see that time it took for each model to run on the CIFAR-10 dataset.

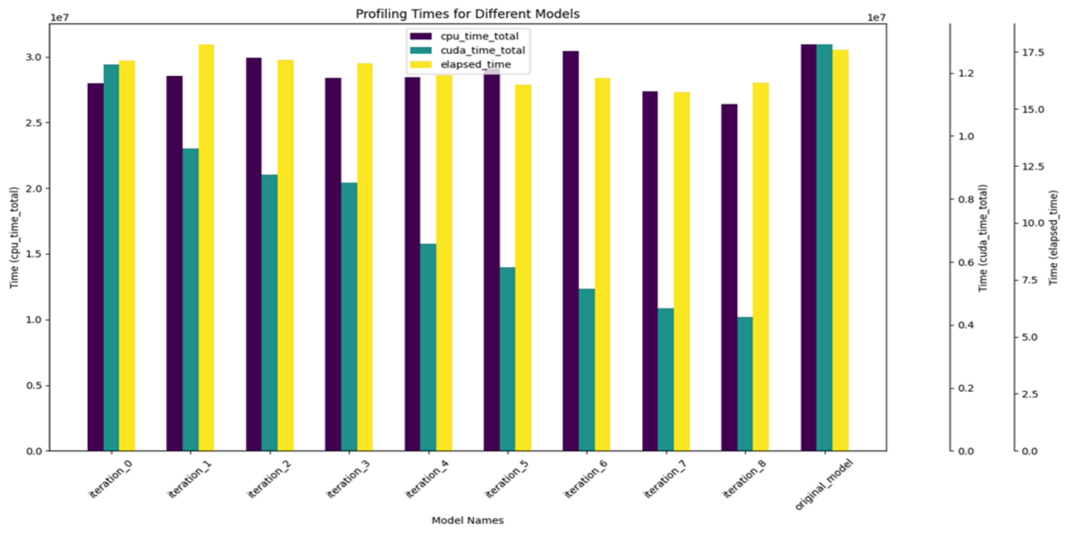


Figure 30*: Measurement of execution time for different iterations (more lightweight in terms of model size) of the DNN models.*

For all the models, the Central Processing Unit (CPU) and Compute Unified Device Architecture (CUDA) time are measured using PyTorch's native method called profiler while the elapsed time is the total time that it took to run, using pythons time package. We can see that the CUDA time, which is the GPU that performs the operations fall-off logarithmically, just like the number of weights and thus floating-point operations. The total and CPU time involve additional procedures such as data loading, which are not tied to the size of the model and thus can't be reduced by compressing the model.

Now we will differentiate the models, not only by size (the iteration) but by training method as well. We have 4 different variants.

- The baselines are the models that were initialized the "compressed" size and trained from scratch
- The knowledge distillation models were produced using KD
- The pruning using pruning, from our own trained model
- The pruned from pre-trained also used pruning, but they were pruned from a pre-trained model found on the internet.

The comparison between the four different variants of the resulting model accuracy when utilizing different iterations is shown in Figure 31.

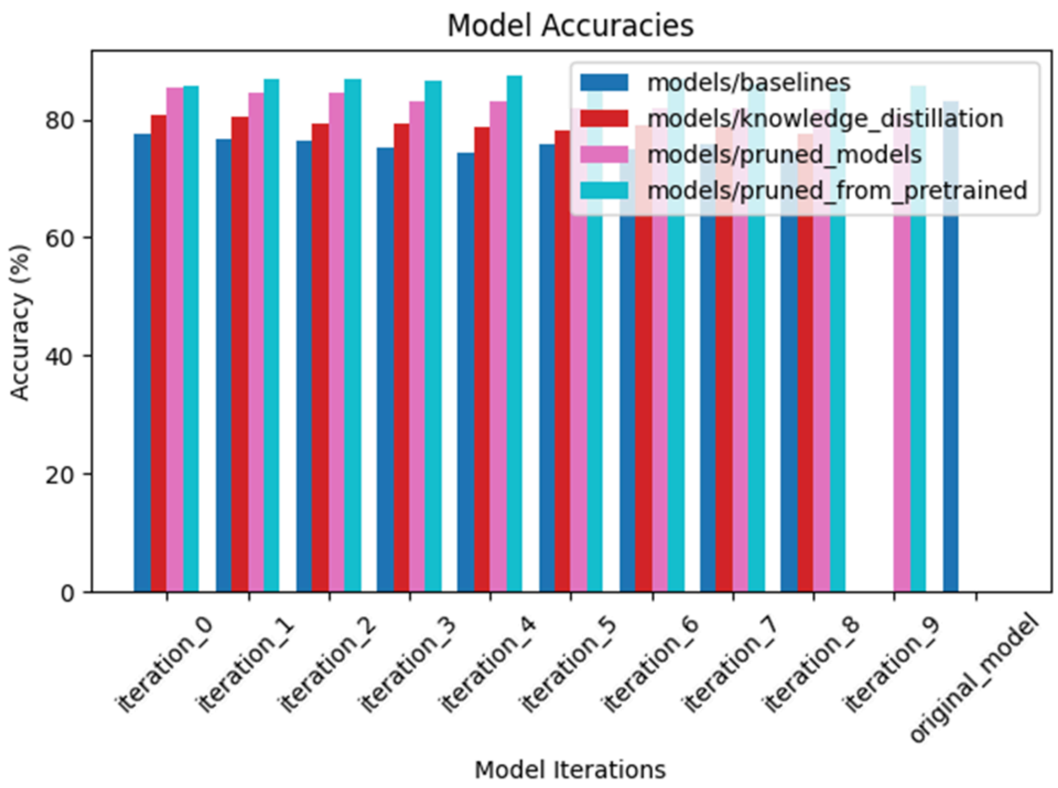


Figure 31*: Comparison between the four different variants of the resulting model accuracy when utilizing different iterations.*

As it is evident, if a pretrained model is available, it should be used as a basis for our compression methods, both having a better performance and saving us the time to train our own. If not, we still are better off training a bigger model and compressing it, rather than training a small one to begin with.

4.4 DEXIT framework: A comprehensive analysis

As aforementioned, a particular tool named DEXIT (Decentralised Early Exit Inference Tool) was developed to support the deployment of the trained EE models in swarm nodes. DEXIT represents a cutting-edge approach to distributed inference in the realm of edge computing and artificial intelligence. By leveraging the power of early exit strategies and decentralised computing, DEXIT addresses several critical challenges in modern AI deployment, particularly in resource-constrained and latency-sensitive environments.

4.4.1 The problem space

In the era of Internet of Things (IoT) and edge computing, traditional centralised cloud-based inference models face several limitations:

1. **Latency**: The round-trip time for sending data to a centralised cloud and receiving results can be prohibitively high for real-time applications.
2. **Bandwidth Constraints**: Transmitting large amounts of data from edge devices to the cloud can overwhelm network capacities.
3. **Privacy Concerns**: Sending sensitive data to centralised servers raises security and privacy issues.
4. **Resource Utilisation**: Cloud-only models often underutilize the computational capabilities of edge devices.
5. **Scalability**: As the number of edge devices grows, centralised models struggle to keep up with the increased load.

DEXIT addresses these challenges through a novel combination of distributed computing, early exit strategies, and peer-to-peer networking:

1. **Distributed Inference**: By distributing the inference process across edge and cloud nodes, DEXIT reduces the burden on any single point in the network.
2. **Early Exit**: Implementing early exit strategies allows simpler inferences to be completed at the edge, reducing latency and bandwidth usage.
3. **P2P Communication**: Utilising libp2p for peer-to-peer networking enables efficient, decentralised data exchange between nodes.
4. **Adaptive Computation**: The framework adapts to the complexity of inputs, allocating more resources only when necessary.
5. **Scalability**: The decentralised nature of DEXIT allows for easy addition or removal of nodes, enabling the network to scale dynamically.

4.4.2 DEXIT High-Level architecture

The DEXIT architecture is designed to be flexible, scalable, and efficient. It comprises several key components that work in concert to enable distributed early exit inference. The workflow of the Dexit process is shown in Figure 32.

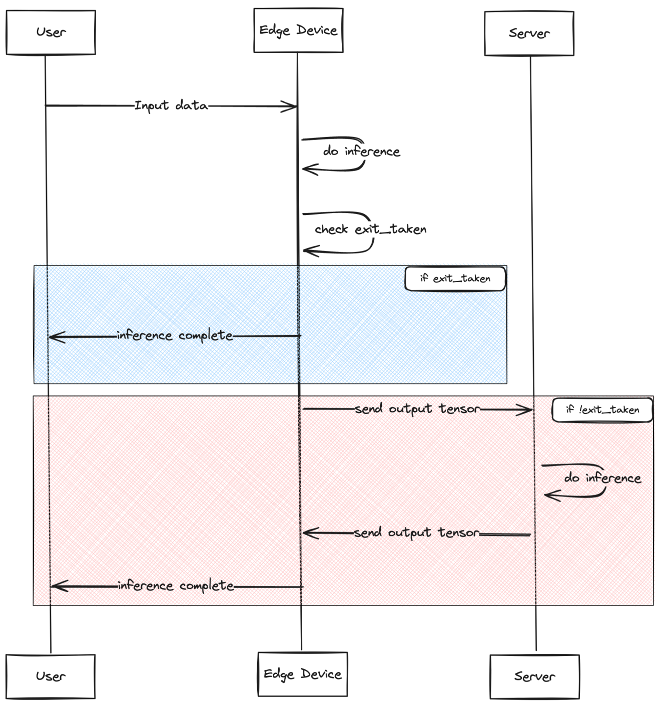


Figure 32*: Workflow of the DEXIT process, illustrating the user (first part of the model), the edge device (mid part of the model) and the server (final part of the model).*

4.4.3 DEXIT Core Architectural Components

4.4.3.1 Edge Device

The Edge Device serves as the entry point for inference requests. It is typically a resource-constrained device (e.g., smartphone, IoT sensor) that initiates the inference process.

Key responsibilities:

- Initial processing of input data
- Making early exit decisions based on confidence thresholds
- Forwarding complex cases to Cloud1 Node

4.4.3.2 Cloud1 Node

The Cloud1 Node acts as an intermediate processing unit with more computational power than the Edge Device.

Key responsibilities:

- Processing more complex inference tasks
- Implementing its own early exit strategy
- Forwarding the most challenging cases to Cloud2 Node

4.4.3.3 Cloud2 Node

The Cloud2 Node represents the final and most powerful computational resource in the DEXIT network.

Key responsibilities:

- Handling the most complex inference tasks
- Providing high-accuracy results for challenging inputs
- Supporting the overall network by processing overflow from other nodes

4.4.3.4 Network layer (libp2p)

The P2P Network Layer, implemented using libp2p, forms the communication backbone of DEXIT.

Key features:

- Decentralised peer discovery
- Efficient message routing between nodes
- Support for various transport protocols
- Network Address Translation (NAT) traversal capabilities

4.4.3.5 Network state management

The Network State Management component keeps track of the overall system state, including peer statuses, inference requests, and results.

Key responsibilities:

- Maintaining a real-time view of the network topology
- Tracking the status of ongoing inference tasks
- Managing the distribution of workload across nodes

The interconnection of these software components is demonstrated in the following schematic, where the decentralized nodes (edge device, cloud1 and cloud2 nodes) are linked between them through the libp2p protocol and the network state management entity keeps track of the overall system state.

```
[Edge Device] <---> [Cloud1 Node] <---> [Cloud2 Node]
       ^                ^                ^
       |                |                |
       v                v                v
[P2P Network Layer (libp2p)]
       ^
       |
       v
[Network State Management]
```

4.4.4 DEXIT Key Software Components

4.4.4.1 P2PHandler (network/handler.py)

The P2PHandler is responsible for managing all P2P network operations within DEXIT.

Key functionalities:

- Network initialization and peer discovery
- Message publishing and subscription
- Direct messaging between peers
- Handling of inference requests and results

Implementation highlights:

```
class P2PHandler:
    def __init__(self, bootnodes, key_path, topic, models, packet_size=1024, 
device='cpu', role=None):
        # Initialize P2P network parameters

    async def init_network(self):
        # Initialize the P2P network

    async def publish_status(self, peer_status: PeerStatus):
        # Publish peer status to the network

    async def send_inference_request(self, peer_id: str, inference_request: 
InferenceRequest):
        # Send inference request to a specific peer

    async def send_inference_result(self, peer_id: str, inference_result: 
InferenceResult):
        # Send inference result to a specific peer
```

4.4.4.2 NetworkState (utils/state.py)

The NetworkState class manages the overall state of the DEXIT network.

Key functionalities:

- Tracking peer statuses
- Managing inference requests and results
- Providing network state summaries

Implementation highlights:

```
class NetworkState:
    def __init__(self):
        self.peer_statuses = {}
        self.inference_results = {}
        self.inference_requests = []

    def update_peer_status(self, peer_status_info: PeerStatus):
        # Update the status of a peer

    def get_peer_by_role(self, role: str) -> str:
        # Get a peer ID based on its role

    def update_inference_result(self, result: InferenceResult):
        # Update an inference result
```

4.4.4.3 CIFARDataLoader (data/dataloaders.py)

The CIFARDataLoader handles data loading and preprocessing for the CIFAR-10 dataset, which is used for testing and demonstration purposes in DEXIT.

Key functionalities:

- Loading and preprocessing CIFAR-10 dataset
- Creating DataLoaders for efficient batch processing
- Supporting customizable sample sizes for testing

Implementation highlights:

```
class CIFARDataLoader:
    def __init__(self, batch_size: int = 4, root_dir: str = './shared/data',
num_samples: int = None):
        # Initialize data loader parameters

    def _create_transform(self) -> transforms.Compose:
        # Create image transformation pipeline

    def _load_dataset(self, train: bool) -> torchvision.datasets.CIFAR10:
        # Load CIFAR-10 dataset

    def get_test_loader(self) -> DataLoader:
        # Get DataLoader for test dataset
```

4.4.4.4 Early Exit Models (early_exit/)

The early exit models are custom neural network architectures that support multiple exit points for inference.

Key features:

- Multiple intermediate classifiers (exit points)
- Confidence thresholds for early termination
- Adaptive computation based on input complexity

The main workflow loop of the DEXIT tool and the published alerts to the subscribed topics of the involved entities are shown in Figure 33.

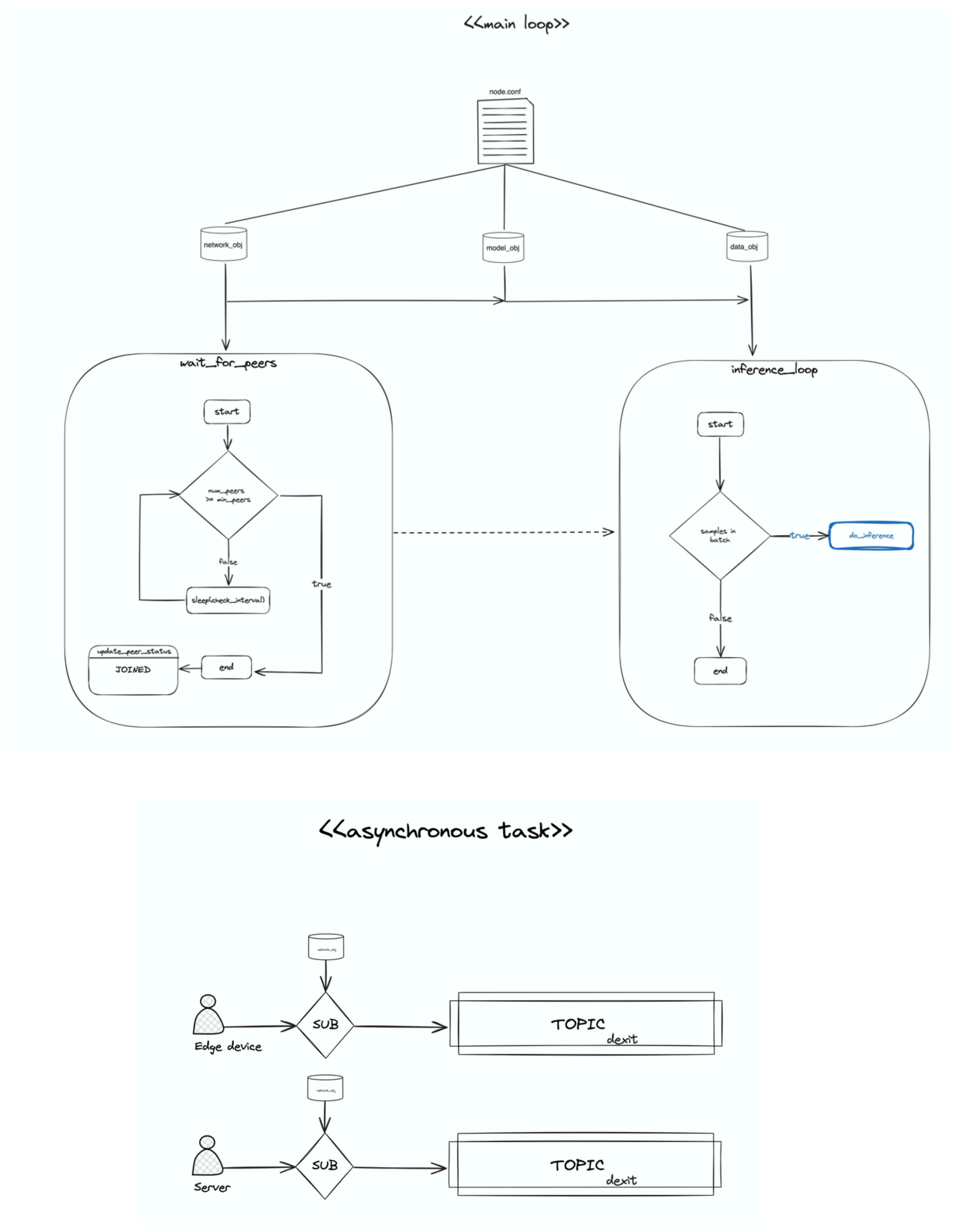


Figure 33*: The main workflow loop of the DEXIT tool (top) and the published alerts to the subscribed topics of the involved entities (bottom).*

The configuration file that can be modified from the user includes three objects: (i) a network object, containing the configurations related to the network in the decentralized architecture, (ii) a model object, containing parameters setting related to the early exit models and (iii) a data object, that handles the inference data. As depicted in Figure 33, only the network object is needed to initiate the wait for peers loop, i.e., each peer waits until the required number of decentralized nodes/peers (configurable by the user) has joined the Dexit framework and the inference process can start. The rest of the inference workflow is described below.

4.4.5 DEXIT Workflow

The DEXIT workflow demonstrates how the system processes inference requests across the distributed network:

1. **Inference Initiation**:
    - An inference request is initiated at the Edge Device.
    - The Edge Device performs initial processing on the input data.
2. **Edge Device Early Exit**:
    - If the confidence threshold is met at the Edge Device, the inference terminates here.
    - The result is returned immediately, minimising latency.
3. **Forwarding to Cloud1**:
    - If the Edge Device cannot meet the confidence threshold, the intermediate result is forwarded to the Cloud1 Node.
    - The P2P network layer handles the efficient transfer of data.
4. **Cloud1 Node Processing**:
    - The Cloud1 Node receives the intermediate result and continues processing.
    - It applies its own early exit strategy based on confidence thresholds.
5. **Cloud1 Early Exit or Forward**:
    - If Cloud1 meets the confidence threshold, it terminates the inference and returns the result.
    - If further processing is needed, the intermediate result is forwarded to the Cloud2 Node.
6. **Cloud2 Node Final Processing**:
    - The Cloud2 Node receives the most complex cases.
    - It performs the final inference steps and returns the result.
7. **Result Propagation**:
    - The final inference result is propagated back through the network to the originating Edge Device.
    - The Network State is updated with the completed inference result.

The inference loop of the DEXIT tool and the send2server functionality when the confidence level has not been reached in the early exits are shown in Figure 34.

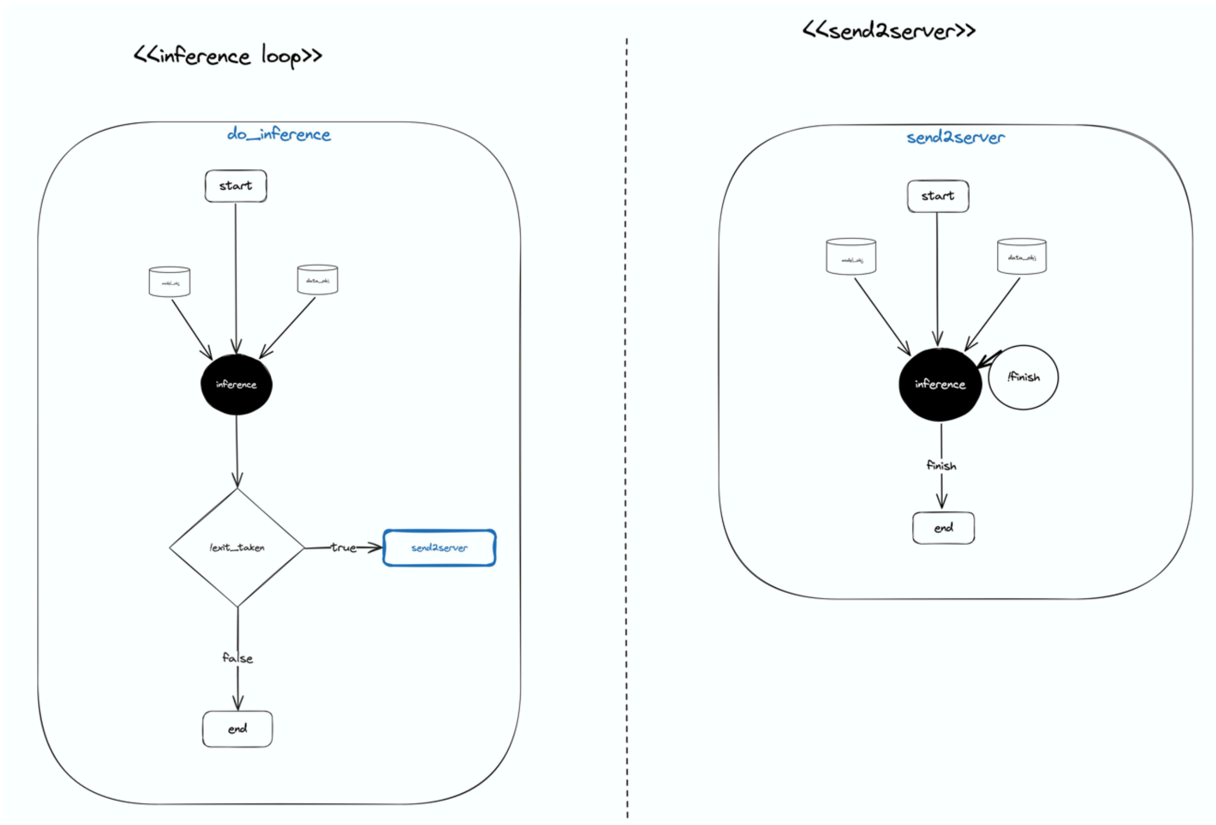


Figure 34*: Inference loop of the DEXIT tool (left) and the send2server functionality (right) when the confidence level has not been reached in the early exits.*

4.4.6 Key Features and Advantages

4.4.6.1 Adaptive Computation

DEXIT's early exit strategy allows for adaptive computation based on input complexity. This results in:

- Reduced overall computational load
- Lower latency for simpler inputs
- Efficient utilisation of resources across the network

4.4.6.2 Decentralised Architecture

The use of a P2P network powered by libp2p provides:

- Improved scalability and resilience
- Reduced dependency on centralised infrastructure
- Dynamic adaptation to network conditions and node availability

4.4.6.3 Flexibility and Heterogeneity

DEXIT accommodates a wide range of devices with varying computational capabilities:

- Leverages both edge and cloud resources effectively
- Adapts to different hardware configurations and constraints
- Supports seamless integration of new devices into the network

Funded by the European Union

4.4.6.4 Reduced Latency and Bandwidth Usage

By processing data closer to the source, when possible, DEXIT achieves:

- Lower end-to-end latency for time-sensitive applications
- Reduced bandwidth consumption, especially beneficial in constrained network environments
- Improved real-time decision-making capabilities

4.4.6.5 Enhanced Privacy and Security

The distributed nature of DEXIT offers inherent privacy and security benefits:

- Sensitive data can be processed locally, reducing exposure
- Decentralised architecture minimises single points of failure
- Potential for implementing federated learning approaches

4.4.7 Challenges and Future Directions

While DEXIT presents a promising approach to distributed inference, several challenges and areas for future research remain:

4.4.7.1 Dynamic Load Balancing

Developing sophisticated algorithms for real-time load balancing across heterogeneous nodes is crucial for optimal performance.

4.4.7.2 Model Consistency and Updates

Ensuring consistency of model versions across distributed nodes and implementing efficient update mechanisms are important challenges to address.

4.4.7.3 Privacy-Preserving Techniques

Integrating advanced privacy-preserving methods, such as differential privacy or secure multi-party computation, could enhance DEXIT's applicability in sensitive domains.

4.4.7.4 Fault Tolerance and Recovery

Implementing robust fault tolerance mechanisms and recovery strategies is essential for maintaining system reliability in dynamic edge environments.

4.4.7.5 Standardisation and Interoperability

Developing standards for decentralised early exit inference and ensuring compatibility with existing edge computing platforms will be crucial for widespread adoption.

4.4.8 Conclusion

DEXIT represents a significant advancement in the field of distributed inference for edge computing. By combining early exit strategies with decentralised computing, it addresses critical challenges in latency, bandwidth utilisation, and scalability. As edge computing and IoT continue to evolve, frameworks like DEXIT will play a crucial role in enabling efficient, real-time AI inference across distributed networks of heterogeneous devices.

The modular and extensible nature of DEXIT opens up numerous possibilities for future enhancements and adaptations to diverse use cases, from smart cities and autonomous vehicles to industrial IoT and beyond. As research in this area progresses, DEXIT stands as a promising foundation for the next generation of distributed AI systems.

4.5 COMMUNICATION-EFFICIENT VERTICAL FEDERATED LEARNING VIA COMPRESSED ERROR FEEDBACK

This section explores innovative approaches to vertical federated learning, focusing on communication efficiency through compressed error feedback [53]. The proposed techniques aim to minimise communication overhead while preserving model accuracy and convergence speed, thus promoting energy-efficient learning processes. The methodology involves compressing the error feedback in federated learning scenarios, thereby reducing data transfer requirements without compromising learning efficacy.

As a natural application of [53], the NOVA group in WP5 is testing the viability of [53] in the space domain. In recent years, vertical federated learning (VFL), which allows different entities to collaboratively train models using their unique feature sets, has emerged as a promising framework for enhancing data-driven applications in Lower Earth Orbit (LEO). Despite this, the communication overhead in VFL remains a significant bottleneck, especially in satellite-based networks. As seen in FedSpace [54], the challenges of connectivity and aggregation in federated learning systems can significantly slow down training processes. More efficient communication and aggregation algorithms are needed, such as those proposed in EFVFL [53]. This work aims to explore the application of VFL in LEO satellite constellations, utilizing state-of-the-art algorithms to address the unique challenges posed by such distributed networks.

5 Positioning of ML/AI tools in TaRDIS

5.1 Overview and relation with TaRDIS requirements

TaRDIS WP2 aims to analyse and review the end-user needs and determine the functional requirements of the TaRDIS development environment. D2.2 [6] synthesised the functional requirements for TaRDIS, where WP5 contributed to D2.2 [6] by identifying the requirements regarding WP5 (as reported in D5.1 [1]). In D2.3 [4], the precise meaning of heterogeneous swarm systems for TaRDIS was defined. It also revealed the details about the necessary structure of the toolbox including the requirements to be fulfilled by each kind of tool included. WP5 contributed to this deliverable, by the definitions of the WP5 tools and the links with the appropriate requirements.

In D2.3 [4] the TaRDIS toolbox architecture was introduced, as shown in Figure 35. The tools on higher levels are built on or use guarantees of tools on lower levels. The positioning of WP5 within this architecture can be identified through 3 units, coloured blue: Intelligent orchestrator, ML inference agents and Federated ML training. The contributions of different tasks in WP5 fit into these architectural units. The tools that belong to each unit are listed in brackets and described in D2.3 [4] in detail, including also their linkages with the TaRDIS requirements, defined in D2.2 [6].

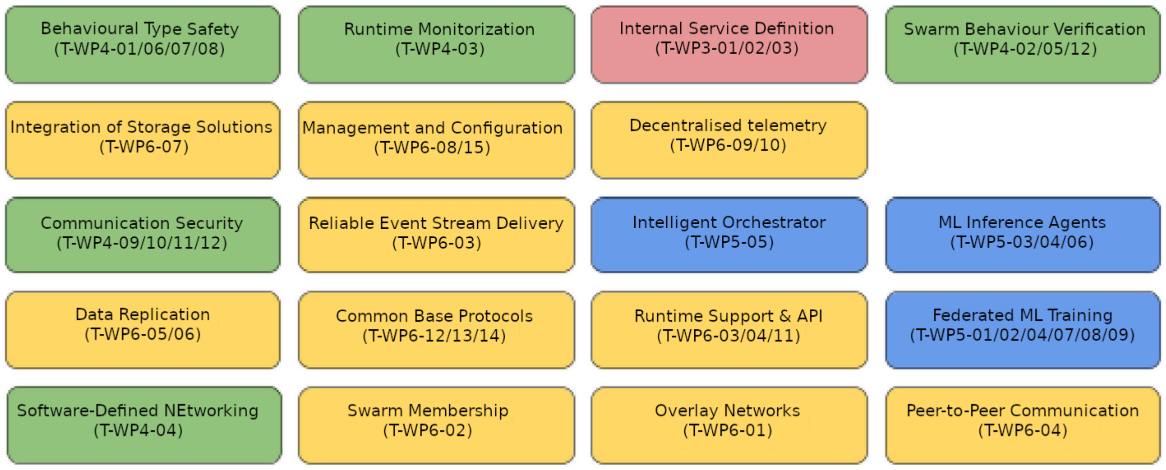

Behavioural Type Safety (T-WP4-01/06/07/08)
Runtime Monitorization (T-WP4-03)
Internal Service Definition (T-WP3-01/02/03)
Swarm Behaviour Verification (T-WP4-02/05/12)
Integration of Storage Solutions (T-WP6-07)
Management and Configuration (T-WP6-08/15)
Decentralised telemetry (T-WP6-09/10)
Communication Security (T-WP4-09/10/11/12)
Reliable Event Stream Delivery (T-WP6-03)
Intelligent Orchestrator (T-WP5-05)
ML Inference Agents (T-WP5-03/04/06)
Data Replication (T-WP6-05/06)
Common Base Protocols (T-WP6-12/13/14)
Runtime Support & API (T-WP6-03/04/11)
Federated ML Training (T-WP5-01/02/04/07/08/09)
Software-Defined NEtworking (T-WP4-04)
Swarm Membership (T-WP6-02)
Overlay Networks (T-WP6-01)
Peer-to-Peer Communication (T-WP6-04)

Figure 35*: The TaRDIS toolbox architecture overview (from D2.3 [4]).*

The Federated ML Training unit includes a set of tools offering solutions for federated learning training: Flower-based FL model training (T-WP5-01), PTB-FLA and MPT-FLA (T-WP5-04), Decentralised Federated Learning Framework - Fedra (T-WP5-09), as well as a tool for data preparation for Flower-based FL model training (T-WP5-02). It also includes two lightweight ML technique tools: Knowledge distillation(T-WP5-07) and Pruning (T-WP5-08). On a higher architectural level, ML inference agents rely on the outputs from the Federate ML Training. Here, we can identify three relevant tools: Flower-based FL model inference and evaluation (T-WP5-03), PTB-FLA and MPT-FLA (T-WP5-04) and Early exit (T-WP5-06). On the same level, under Intelligent orchestrator, the Federated AI Network Orchestrator - FAuNO (T-WP5-05) is defined. This architectural view pervades the results of the different WP5 tasks.

5.2 Interaction with TaRDIS programming abstractions

TaRDIS WP3 aims to specify the programming model and the APIs (Application Programmer Interfaces) that can be offered to programmers by the TaRDIS toolbox. WP3 is also focusing on integrating a set of tools on an IDE (Integrated Development Environment) that simplifies the development of decentralised applications deployed in diverse settings. A first outline of the TaRDIS programming model and APIs is provided in D3.1 [55] (led by WP3), where WP5 has contributed to D3.1 [55] by providing initial definitions of WP5 APIs (as reported in D5.1 [1]). D3.2 [5] provided a detailed evaluation of the TaRDIS IDE platform, where WP5 created descriptions of its proposed candidate applications, supported by mock-ups, screenshots and detailed explanations. D3.3 [56] presents the second revision of the TaRDIS programming models and TaRDIS toolkit APIs, and WP5 contributed to this deliverable by providing updates on WP5 APIs defined in D3.1 [55]. Also, the TaRDIS models and APIs documentation is currently available under the TaRDIS wiki, where WP5 provided an overview of its APIs. The WP5-related Application Programmer Interfaces (API) definitions are aligned with the proposed WP5 tools. The proposed WP5 tools will be integrated in the TaRDIS IDE platform during the last year of the project, as drafted in D3.2.

5.3 Interaction with TaRDIS property verification tools

The work of WP4 is focused on developing formal analyses to establish the soundness, security, and reliability of a heterogeneous swarm. The first step was to identify desirable properties for analysis, which were reported in D4.1 [57]. In this stage, WP5 and WP4 have worked together on identifying several desirable FL properties (FL roles of agents, FL data privacy, FL message delivery, and FL clients' equality). In D4.2 [58], WP4 has reported results on formal verification of the correctness of centralised and decentralised FL algorithms developed in the PTB-FLA tool (T-WP5-04). The correctness of two generic FL algorithms was verified by proving two properties: deadlock freedom and successful Federated Learning Algorithm (FLA) termination. The properties have been formalised in communicating sequential processes calculus (CSP) and verified in the Process Analysis Toolkit (PAT) [59]. In addition, a systematic approach to translating Python code that follows a restricted actor-based programming model to a corresponding CSP model is developed. In future work, this approach should serve as a basis for developing a tool for the automatic translation of certain classes of Python code to CSP models.

5.4 Interaction with TaRDIS data management and distribution primitives

TaRDIS WP6 aims to provide communication, membership and data management primitives. As stated in D5.1 [1], the connection between WP5 and WP6 aims to provide the ability to combine ML with decision making regarding resource availability.

The results accomplished within WP6 provide that designed services are able to collect, aggregate, transform and store diverse monitoring-related data from heterogeneous sources and with respect to resource availability. The metrics are collected from a few places: (i) node metrics, (ii) applications (in containers) metrics, and (3) other places in the toolbox that provide metrics such as distributed protocols and probing mechanisms. All metrics are stored, and therefore available for others in OpenMetrics format. A unified interface of the system and application state progression over time is accessible to users. This will allow for informed decision making. The toolbox also offers an API for machine learning models to be both trained on and put in the role of decision makers, which provides the automation of the decision process. The telemetry data is being stored by the toolbox in time series manner (i.e. data points with time). The telemetry data means, for instance, CPU load, disk usage, memory usage, network properties... This data is accessible by ML models through a specifically designed API directed towards their needs. An agent that requires data from the platform for training, simply needs to send a point in time when the last contact occurred. If the agent for whatever reason misses to get data in e.g., regular intervals, there is an interface to get metrics in a specific period between two dates. This ensures that machine learning models can be trained at their own pace. Even if an agent crashes, or goes offline for some period of time, newly added data points cannot be missed or skipped.

Task 5.2 (from WP5) is responsible for automated analysis and planning, while WP6 will perform execution and monitoring, according to the framework: Monitor, Analyse, Plan, Execute, and Knowledge (MAPE-K) [60]. The list of federated ML algorithms that need to be provided by the TaRDIS toolkit, in the context of WP6 needs include three categories:

- supervised learning algorithms, for regression and classification tasks, as well as for time-series forecasting, for instance
- unsupervised learning algorithms, as anomaly detection tasks and
- reinforcement learning algorithms, e.g., decision-making for resource optimization.

The most important link with WP6 is the capture of network and node usage metrics to be fed to the orchestration RL agent (or agents), and the communication between agents in the deployment phase (the details are available in the requirements Deliverable D2.2[6]).

5.5 Interaction with TaRDIS implementation and evaluation

TaRDIS WP7 is focused on evaluation activities concerning the technical achievements of the project. D7.2 [61] represents a report on preliminary validation of the toolbox. WP5 contributed to D7.2 [61], by providing descriptions for the WP5 tools and connecting them to evaluations on TaRDIS use cases, as well as to the relevant KPIs. In specific, sample ML modelling applications have been designed and developed for each use case separately, showcasing the implementation of different ML algorithms for the diverse requirements of the TaRDIS use cases (analysed in Section 6). Apart from the ML application analysis, WP7 has identified the particular tools developed in the framework WP5 that will be used for demonstration and validation purposes:

- *Distributed navigation concepts for LEO satellites constellations*: For the GMV use case, the ML tools from WP5 that will be utilized in the demonstration for performing the federated learning of at least two ML models (one model based on supervised learning and one model based on reinforcement learning) are the T-WP5-04 PTB-FLA, T-WP5-09 Fedra or T-WP5-01 Flower-based tool and T-WP5-05 FAuNO. In addition, the tools offering lightweight functionalities in the inference process will be used, i.e., T-WP5-07 Knowledge Distillation or T-WP5-08 Pruning tools, to save computational resources.
- *Multi-Level Grid Balancing*: For the EDP use case, the T-WP5-09 Fedra tool (or the T-WP5-01 Flower-based tool) will be utilized to perform the FL training of two types of ML models (based on supervised learning and reinforcement learning). Moreover, the functionalities offered by the T-WP5-08 Pruning tool will be utilized to provide more lightweight ML model inference at the edge devices, without significant degradation of their accuracy.
- *Privacy-Preserving Learning Through Decentralized Training in Smart Homes*: The TID use case inherently uses the T-WP5-10 Federated Learning as a Service (FLaaS) tool in order to demonstrate the privacy-preserving federated learning framework in a decentralized configuration. Furthermore, this use case will also utilize the energy-efficient APIs provided by WP5 at the level of the FLaaS app and specifically the ones adopted for FL clients. In specific, at least one of the lightweight

functionalities for inference will be demonstrated upon compatibility, i.e., T-WP5-06 Early-Exit, T-WP5-07 Knowledge Distillation or T-WP5-08 Pruning.

- *Highly resilient factory shop floor digitalisation*: For the ACT use case demonstration, WP5 tools will be involved in the Machine app, the Transport app, the Manager app and the Event Archive App. In this context, Flower-based tool T-WP5-03 will be utilized for inference of ML models, as well as evaluation of their accuracy. In addition, the models will be trained using the T-WP5-01 Flower-based tool, while the T-WP5-02 Data preparation for Flower-based FL model training tool will be used to pre-process the dataset and overcome potential irregularities.

5.6 Interaction with TaRDIS dissemination, exploitation and standardisation

TaRDIS WP8 aims to monitor and collect the project results and outcomes. The Energy use case has received a strong contribution from WP5 regarding Renewable Energy generation and Energy consumption forecasts using Federated Learning Model, developed by University of Athens and has been already included in the use case's swarm code. Additionally, a poster titled "*Collaborative Intelligence Sharing in Energy Communities via Federated Learning*" was recently accepted for the ETSI Artificial Intelligence Conference - How Standardization is Shaping the Future of AI [62].

6 ML MODELLING OF TARDIS USE CASES

6.1 ACT USE CASE

The initial Machine Learning model for the ACT use case, as well as the underlying use case scenario description, is provided in D5.1 [1]. The implementation utilises unsupervised learning approaches (due to the lack of ground-truth labels for this use case). It represents an integration of two ML methodologies: K-means clustering algorithm and representational learning through autoencoders. In this section, we provide experimental results described ahead; further innovations regarding the machine learning modeling of the use case were detailed in Section 2.1.3 of this deliverable.

6.1.1 AUTOENCODER-BASED OUTLIER DETECTION AND K-MEANS

We train the autoencoder (using the TensorFlow framework) on a subset of a dataset, where the instances are non-anomalous. The classification of the test data is relying on a threshold value that is dependent on the dataset structure. Following the training phase, the test data is classified as either normal or anomalous based on a threshold value that depends on the structure of the testing dataset. In our experiments, the testing dataset includes normal and anomalous data, as in Experiment 1, and additionally comprises pseudo-normal and pseudo-anomalous data, as described in Experiment 2. In the first experiment, an instance in the testing dataset is classified as an outlier if its reconstruction error exceeds a predefined threshold, determined as the maximum reconstruction error observed during the training phase. In the second experiment, the threshold is set based on the 95th percentile of the reconstruction error values from the training data, also established during the training phase.

The outliers detected by the autoencoder are then verified against those identified by the two K-means algorithms utilised in our approach. We refer to these approaches as Algorithm 1 and Algorithm 2, for simplicity.

6.1.1.1 Algorithm 1

The hybrid outlier detection method utilises the K-means clustering algorithm ([15], [16]) to identify a wide range of potential outliers in the dataset. After K-means identifies clusters, we analyse these clusters to detect outliers. As already explained, typically, potential outliers are found in small clusters that are significantly distant from large clusters. Our implementation utilises the K-means algorithm from the scikit-learn Python library [17].

6.1.1.2 Algorithm 2

Due to the need for a detailed comparison between outliers detected by the autoencoder and those detected by the K-means algorithm, we introduced another algorithm into the comparison phase. This algorithm is designed to flag a data point as an outlier if it is significantly distant from all centroids. Initially, we identified the optimal clustering configuration using only non-anomalous data, as dictated by the structure of our training dataset. The optimal number of clusters was identified using the Silhouette score. Subsequently, we introduced points from the testing set, and for each added point, we computed its distance from the centroids. Each point in the test set was evaluated to determine whether its distance exceeded the threshold, thereby designating it as an outlier, or if its distance was lower, assigning it to one of the existing clusters. This threshold value was established based on the 95th percentile of distances calculated from the training set.

6.1.2 The experimental dataset

Due to the current unavailability of real data provided by the use case provider, we carried out a significant study for identification of a suitable publicly available dataset. This task proved to be quite challenging. Most datasets accessible online are either insufficiently large or not designed for use in anomaly detection or classification tasks. After extensive research, we selected the MetroPT-3 Dataset [63] for its relevance and comprehensive data collection, which aligns well with our anomaly detection objective.

The dataset, known as the MetroPT-3 Dataset, was gathered from a metro train in an operational environment. It includes measurements of pressure, temperature, motor current, and air intake valves from a compressor's Air Production Unit (APU). This dataset addresses real-world predictive maintenance challenges and is applicable for tasks such as failure prediction and anomaly detection. It contains multivariate time series data from various analogue and digital sensors on a train's compressor. These data recorded the temporal behaviour and fault events of the industrial equipment over a period of seven months. Further details on the data collection process and potential compressor system failures are provided in the accompanying papers [64, 65].

6.1.3 Results and discussion

This section presents the results of two experiments conducted on a subset of the original dataset. The subset consists of 6000 instances and 14 features. The nature of the original dataset is such that, even though the data is not labelled, there is a guideline ([63]), that indicates the time intervals during which "Air Leak" occurs, thus identifying 1.975% of the data as anomalies. This method yields a subset of the dataset with 29,960 instances representing anomalies. All remaining data is classified as normal, resulting in a dataset with 1,516,948 instances of non-anomalous data.

6.1.3.1 Results on Experiment 1

The dataset used for the first experiment consists of 4800 instances with 14 features in the training set and 1200 instances with 14 features in the test set. The test set contains 600 outliers, representing 50 % of the total data points inside the test set.

Upon applying the autoencoder to these training and test sets, and cross-checking its results with K-means, 531 final anomalies are detected, which constitutes 88.5% of the total anomalies, as shown in Table 1. When comparing these results with the ground truth (Table 2), the precision achieved is 99.81%, recall is 88.33%, accuracy is 94.08%, and the F1 score is 93.72%. The first image below (Figure 36), shows the training loss across epochs, while the second image (Figure 37) presents the training time relative to the number of data points used for training the autoencoder.

Table 1: Final results on Experiment 1, for anomaly detection

Outliers total	Outliers detected	[% od anomalies]
600	531	88.5000

Table 2: Final results on Experiment 1 versus Ground truth, for anomaly detection

Precision	Recall	Accuracy	F1 score
0.9981	0.8833	0.9408	0.9372

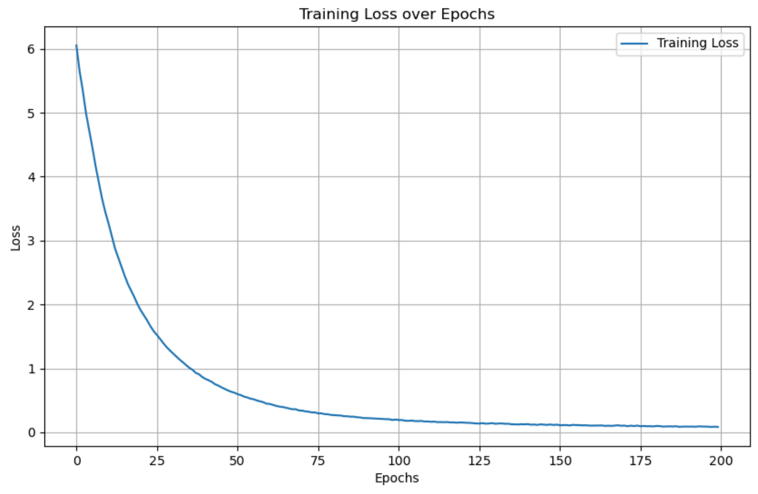


Figure 36*: Training Loss of AE over Epochs.*

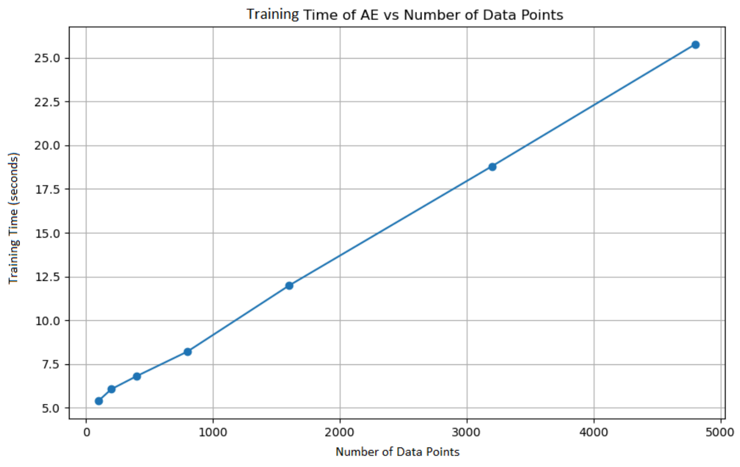


Figure 37*: Training Time of AE versus Number of Data Points.*

6.1.3.2 Results on Experiment 2

In the next experiment, we introduced label flipping, by randomly selecting instances within a subset of the dataset: a certain percent of the known non-anomalous data was relabelled as anomalies, and similarly, the same percent of the anomalous data points were relabelled as normal (non-anomalies). We then repeated the testing on using the autoencoder and different versions of the K-means algorithm. In our evaluation, we tested both the incorrect (flipped labels) and correct (true ground truth) datasets. Our aim was to show how robust this method is against errors in labelling.

In this experiment, we maintained the same training and test set proportions as in the first experiment: 80 % for training and 20 % for testing. To make the task more challenging for our method, 10% of the training data were mislabelled as non-anomalies (when they were actually anomalies), and the test set included 600 true anomalies along with 10% inaccurately labelled as anomalies (non-anomalous data falsely presented as anomalous data). Despite these challenges, the method performed robustly, achieving slightly lower but still high-performance metrics compared to the true labels. In this case, after applying the autoencoder to the training and test sets, and cross-verifying its results with K-means, a total of 551 anomalies were detected, representing 91.8% of all identified anomalies, as presented in Table 3. In comparison with the ground truth (Table 4), the model achieved a precision of 96.19%, recall of 73.61%, accuracy of 82.41%, and an F1 score of 83.40%.

Table 3: Final results on Experiment 2, for anomaly detection

Outliers total	Outliers detected	[% of anomalies]
600	551	91.8333

Table 4: Final results versus Ground Truth, for anomaly detection

Precision	Recall	Accuracy	F1 score
0.9619	0.7361	0.8241	0.8340

To further validate the method's robustness, we increased the proportion of inaccurate labels from 20% to 90% in increments. The relative deterioration of the F1 score correspondingly increased with the percentage of inaccuracies, reaching a maximum of 0.23 when 90% of the labels were flipped in the training dataset, as shown in Figure 38. This experiment demonstrates that the method is robust when applied in real-world scenarios where training data may be affected by label noise, which can result from imperfect labeling processes or the presence of anomalies in the training dataset.

6.1.3.3 Scalability and computational aspects

The method employed is beneficial for larger datasets despite their computational demands. The provided Figure (Figure 39) illustrates the linear increase in execution time relative to the dataset size.

Additionally, in the case of using a larger number of features, it is suggested to employ an efficient method for dimensionality reduction that utilises the variational autoencoder approach, as described in [66]. In our ongoing efforts to achieve even better results, we intend to investigate the potential of transformer models for anomaly detection, as proposed in [67].

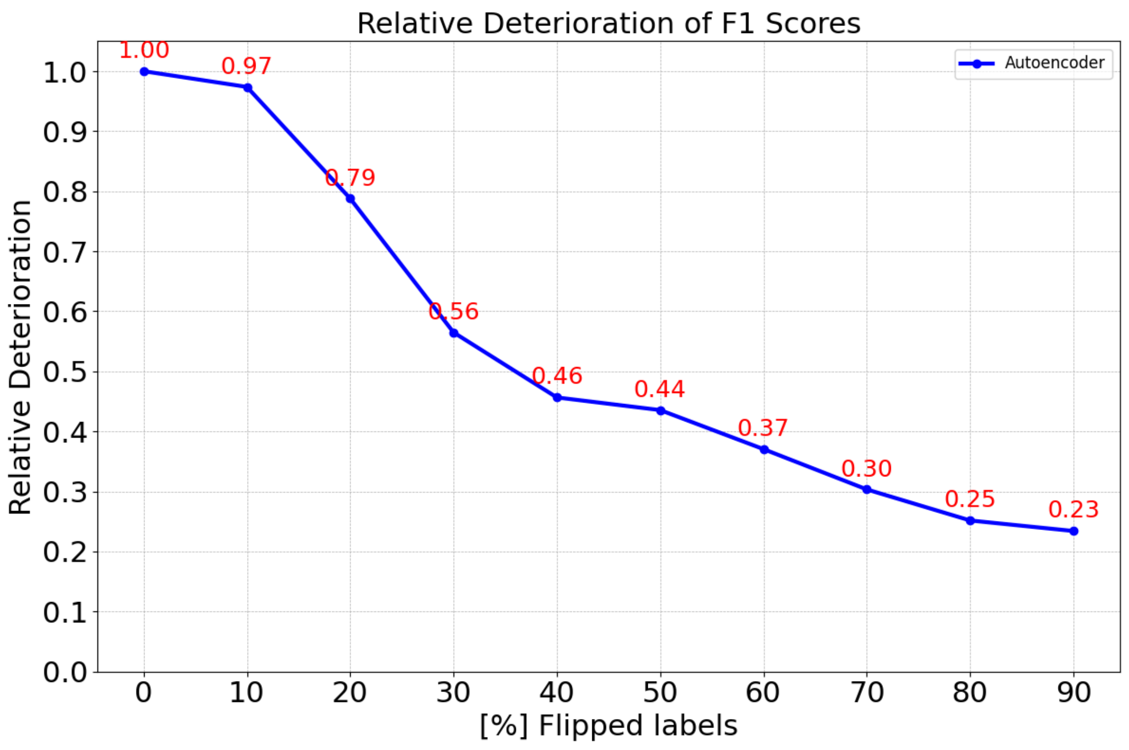


Figure 38*: Relative deterioration in F1 score with varying number of inaccurate labels.*

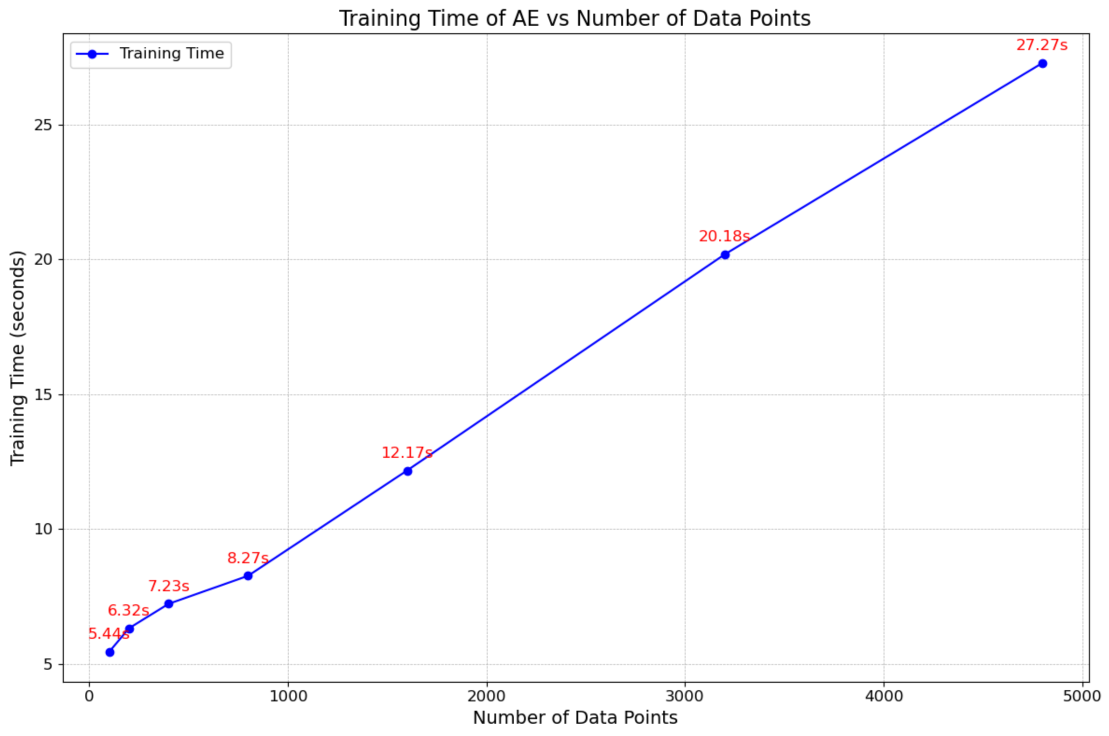


Figure 39: *Scalability of the Autoencoder for the ACT use case.*

6.2 GMV USE CASE

A centralised fast and precise model for Orbit Determination using machine learning models and Neural Networks [68] was developed, which is planned to be decentralised in a later stage of the project.

The GMV use case with respect to WP5 addresses the application of machine learning for precise and efficient orbit determination in Low Earth Orbit (LEO). As outlined in the paper "Precise and Efficient Orbit Prediction in LEO with Machine Learning using Exogenous Variables" [68], traditional methods such as Simplified General Perturbations 4 (SGP4) struggle with non-conservative forces like atmospheric drag and gravitational perturbations. This paper proposes a machine learning algorithm that utilises past positions and environmental variables, significantly reducing computational costs while improving prediction accuracy. The methodology integrates exogenous variables to capture the effects of non-conservative forces and applies time-series techniques for forecasting, achieving low positioning errors and enhancing Space Situational Awareness (SSA) capabilities. Figure 40 shows the proposed network architecture. It represents a two-layer Model Architecture, i.e. an integration of two independently trained models, where one serves partially as the input source for the other.

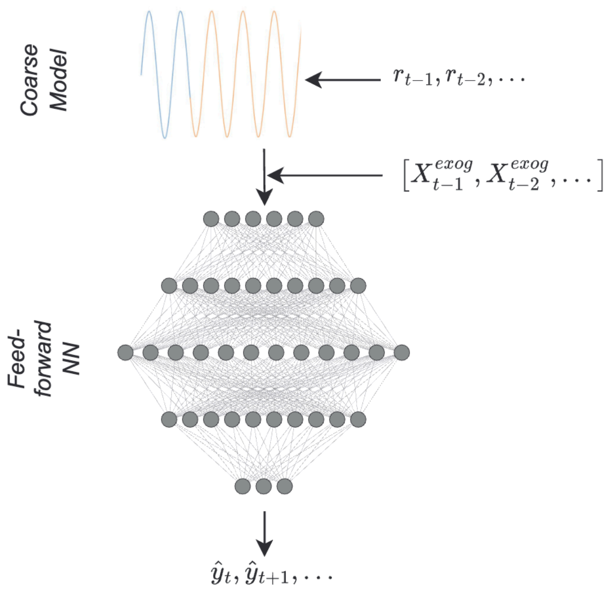

Coarse Model
$r_{t-1}, r_{t-2}, \ldots$
$[X^{exog}_{t-1}, X^{exog}_{t-2}, \ldots]$
Feed-forward NN
$\hat{y}_t, \hat{y}_{t+1}, \ldots$

Figure 40*: Two-layer Model Architecture.*

To address these challenges, the proposed machine learning algorithm leverages historical position data and environmental variables such as atmospheric density. The use of time-series techniques and exogenous variables enables the model to achieve low positioning errors at a reduced computational cost, significantly enhancing the SSA capabilities by providing faster and reliable orbit determination for an increasing number of space objects. In addition, the survey "Machine Learning in Orbit Estimation" provides a comprehensive overview of the state-of-the-art in applying machine learning for orbit determination, orbit prediction, and atmospheric density modelling. The survey discusses the limitations of traditional methods and emphasises the potential of machine learning to improve accuracy by deriving unmeasured object characteristics and enhancing the modelling of non-conservative forces [69].

The next stage of the project will involve integrating Physics-Informed Neural Networks (PINNs) and Neural Ordinary Differential Equations (NeuralODEs) to enhance the model's accuracy and generalizability.

PINNs introduce physical laws as constraints during the training process by incorporating a physics-based loss function in addition to the MSE loss between data points. This guarantees that the model's predictions are consistent with known physical principles, such as the effects of gravitational forces and atmospheric drag, while also improving generalization to new scenarios. By enforcing these laws, PINNs ensure physically sound results, and also reduces the need for larger datasets to cover unseen conditions. PINNs have been proven effective in solving forward and inverse problems involving nonlinear partial differential equations, as demonstrated by [70]. In the same realm of PINNs there is a new layer architecture called NeuralODE, which embeds differential equations directly into the architecture of the machine learning model. This allows the propagation process to be modelled to be constrained/controlled by orbital mechanics during both training and inference, by having differential equations serve as layers within the network. This approach is particularly promising for improving traditional orbit propagation methods, as discussed by [71]. Next steps will also involve the training of a machine learning algorithm to perform decentralized Orbit Determination and Time Synchronization utilizing inter-satellite and satellite-ground stations measurements, as well as the training of a reinforcement learning model for the Inter Satellite Link scheduling optimization and reconfiguration.

6.3 EDP USE CASE

As described in D5.1 [1], the use case is based on a realistic mathematical model that considers various aspects of a modern smart home environment. Specifically, the model accounts for energy generated by renewable sources, household energy demands, an energy storage system (ESS), market-dependent electricity prices, and both indoor and ambient temperatures. The presented approach aims to minimize the energy cost required for non-controllable energy consumption and the operation of the controllable Heating, ventilation and Air-Conditioning (HVAC) system by promoting self-sufficient satisfaction of smart home energy demands using renewable energy.

During the second year of the TaRDIS project, the LSTM models were developed that will be integrated into our method to forecast smart home energy consumption and renewable energy generation based on historical data. These predictions consider time-varying parameters, such as solar panel energy production over time. The LSTM models are trained on data from individual smart homes, tailored to specific areas (e.g., ambient temperature or solar energy generation) and the energy habits of residents (e.g., increased energy demands when working remotely).

Moreover, the designed Deep Reinforcement Learning (DRL) algorithm, based on the Deep Deterministic Policy Gradient (DDPG) actor-critic model, as described in D5.1, optimizes the energy provision to the HVAC system and the charging/discharging of the ESS. The DDPG model observes the dynamic state of the problem, including the LSTM outputs (predicted energy generation and requested energy for load activation), which describe future states of the environment. The optimization objective is to maintain a comfortable indoor temperature while minimizing energy costs.

Following the mathematical formulation of the energy use case reported in the previous deliverable D5.1, we incorporated the forecasting functionality in the Home Energy Management System (HEMS). The environment state is now:

- renewable generation output
- non shiftable power demand
- Energy Storage System energy level
- Outdoor temperature
- Indoor temperature
- Buying electricity price
- Time slot index in a day
- Non-shiftable power demand values predicted by the forecasting LSTM model for future time instance
- renewable generation output values predicted by the forecasting LSTM model for future time instance

Moreover, we have made the DRL (DDPG model) more sophisticated, including trade-off rewards (temperature comfort vs energy cost). The Home Energy Management System (HEMS) that observes the smart home environment, acts on it and receives a reward based on the optimization objective is shown in Figure 41.

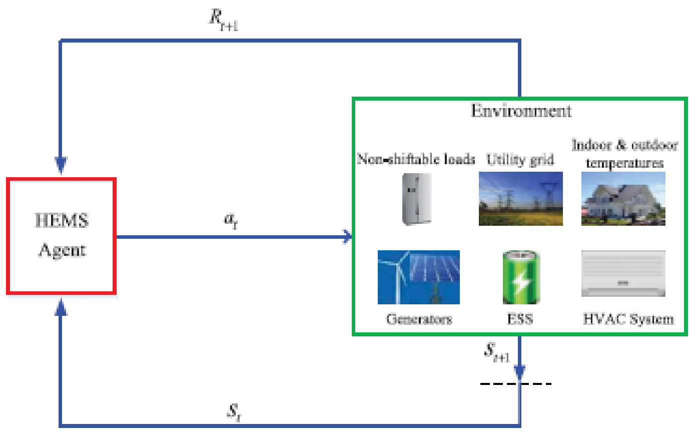


Figure 41*: The Home Energy Management System (HEMS) that observes the smart home environment, acts on it and receives a reward based on the optimization objective.*

To this end, we have trained an LSTM network for forecasting the power production from solar panels using the southern Germany dataset time series (Figure 42). For training data, we used all-year data except April and for testing the data for the month of April. The fine-tuned model achieves mean error = 0.22 kWh, following the time-domain trends.

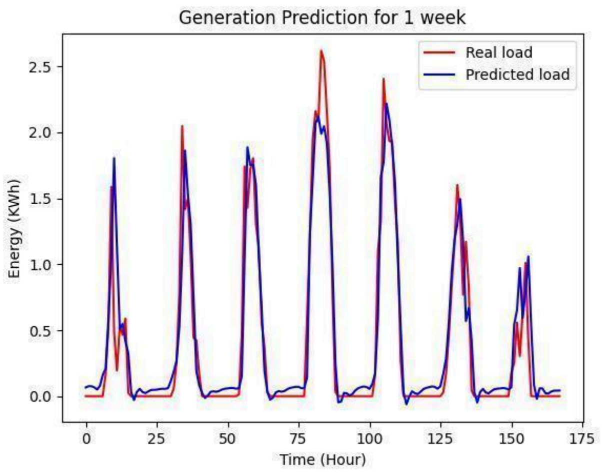


Figure 42: *Forecast of generation power from solar panels of the LSTM model compared to the real data for 1 week of April.*

In a similar manner, we have trained an LSTM network for forecasting the power consumption of the smart home, using the southern Germany dataset time series (Figure 43). For training data, we used all year except April and for testing the data gathered for the month of April. The fine-tuned model achieves Mean error = 0.14 kWh, following the time-domain trends. It should be noted that this model achieves more challenging prediction, since the consumption requests are more irregular.

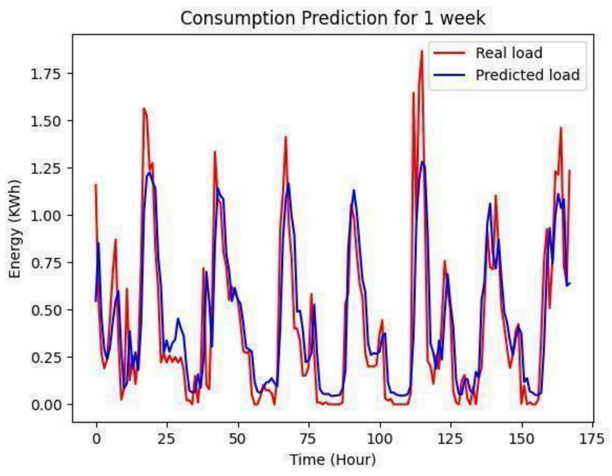


Figure 43: *Forecast of power consumption requests of the smart home using the trained LSTM model compared to the real data for one week of April.*

The above forecasting model outputs are also used to train a DRL (DDPG) model with varying β, i.e., trade-off between temperature comfort 19.5 < Inside Temperature < 22 and energy cost (ESS depreciation + cost of the HVAC input power), see Figure 44. As depicted, the DDPG model stabilises the time-domain temperature regardless of ambient temperature and solar panel produced power, providing the required input power to the HVAC to keep the temperature within the comfort bounds. The DDPG model draws energy from the grid when required to fulfil the consumption demands β = 2 (energy cost is important).

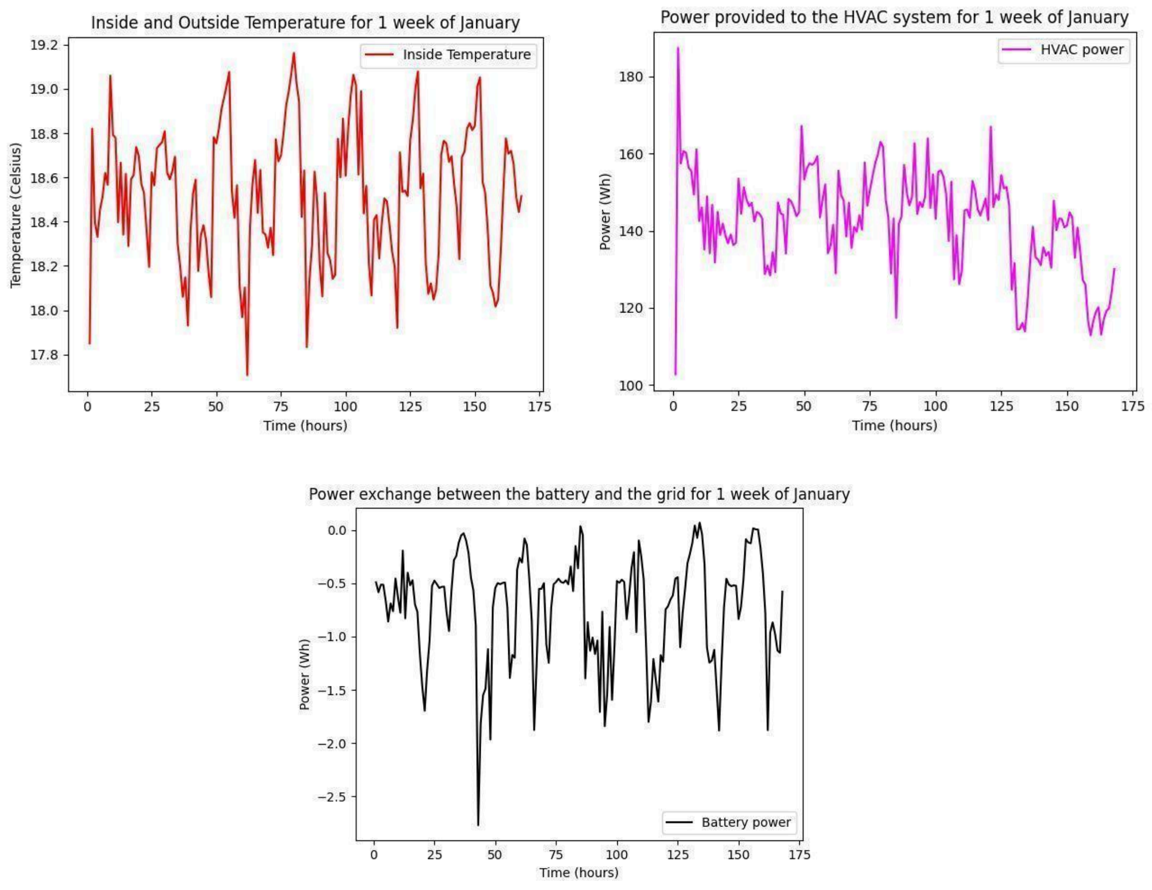


Figure 44*: Inside temperature (upper left), power provided to the HVAC system (upper right) and power exchange between the ESS and the grid (below) using the trained DDPG model for 1 week of April.*

Moreover, we have produced several results for different scenarios, i.e., for different trade-off index (temperature conform vs energy cost) and for different months (January vs July). The results are shown in Figure 45.

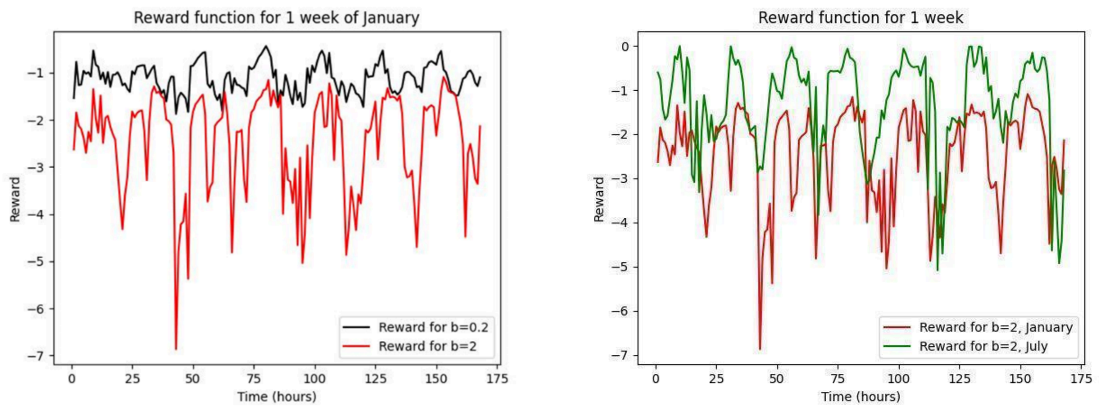


Figure 45*: Reward function of the DDPG model for different β -energy cost vs temperature comfort trade-off- (left) and for different months -January and July- (right).*

The developed method and DDPG model with LSTM assistance can be further extended to multiple smart homes that exchange individual energy surpluses from renewable sources to achieve a zero-sum energy footprint. This approach promotes the use of renewable energy within a smart home and an energy community of multiple homes, meeting specific energy needs with locally produced renewable energy and minimizing grid utilization and associated costs.

6.4 TID USE CASE

TID's use case concerns intelligent homes where different devices are part of an automated system that is designed to facilitate everyday life. In particular, these devices support ML-based functionalities that require the training of neural network (NN) models in a collaborative and distributed way among these devices. FL [72] has been designed for this purpose. However, it requires the devices (clients) to train the models locally. However, as models become more and more complex, devices (e.g., handheld, Internet-of-Things (IoT)) might not be able to participate in the training or might slow down the training. In previous deliverables (D2.2 and D5.1), we discussed how the heterogeneity (in terms of computation, energy, memory) of these devices can be overcome through the Split Learning (SL) paradigm. In this deliverable, we give updates on ongoing research efforts on this aspect with the goal of minimizing the maximum training time among all clients in a system[1], which is called makespan. This is part of an ongoing journal submission that extends a previous article [73] (developed outside of TaRDIS) towards a more generic setting, as described below.

In SL, clients offload a part of their training task to a helper, which could be, for example, a Virtual Machine (VM) on the cloud or a lightweight container in a base station beyond 5G networks. Formally, an NN comprising L layers1 (1, . . . ,L) is split into three parts (part-1, part-2, and part-3) of consecutive layers ([1, . . . , σ1], [σ1 + 1, . . . , σ2], [σ2 + 1, . . . ,L]) using 2 cut layers {σ1, σ2} ∈ Σ. Then, part-1 and part-3 are processed at the clients, and part-2 at the helper. This allows the resource-constrained clients to offload computationally demanding processes to the helper while keeping their data locally.

In this deliverable, we consider the setting of **Hybrid Federated and Split Learning (HFSL)** to manage both resource-constrained clients and more powerful ones (capable of supporting on-device training). This is similar in spirit to works in literature [74]. In this setup, clients may train the NN model either through SL in collaboration with a helper or through FL, i.e., train the entire NN model locally. The motivation for the hybrid approach is that clients with higher-end devices may train through FL without increasing the makespan. Consequently, this will reduce the load and alleviate the resource demands at the helpers which will assist only the resource-constrained clients. The figure below (Figure 46) depicts this setting.

[1] In particular, we focus on the training time of a single batch of input data, leveraging the structural nature of the training process (that consists of multiple batch processings/updates throughout the local epochs and training rounds).

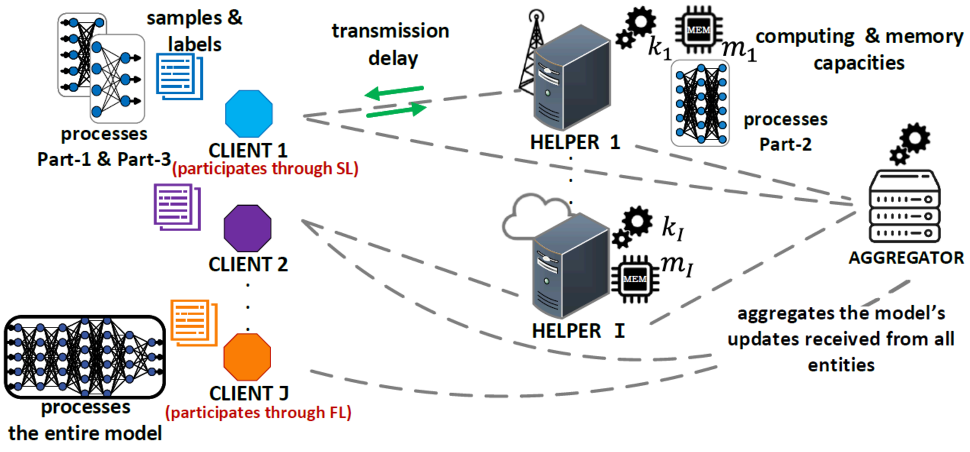

samples & labels
processes Part-1 & Part-3
CLIENT 1
(participates through SL)
transmission delay
HELPER 1
k_1
m_1
computing & memory capacities
processes Part-2
AGGREGATOR
CLIENT 2
HELPER I
k_I
m_I
aggregates the model's updates received from all entities
CLIENT J
(participates through FL)
processes the entire model

Figure 46*: The hybrid federated and split learning setting considered, the considered network topology, its resources, and the processing tasks per entity. Some clients, e.g., client 1, participate in the training through parallel SL, by offloading part-2 to a helper, while others, e.g., client J, participate through FL and perform on-device training.*

Driven by the time measurements of a testbed that are available online [75] and the heterogeneity even among devices of similar capabilities, we identify three key decisions:

1. the training protocol employed by each client (FL or SL);
2. the client-helper assignment which is tied to the helpers' memory and computing capacities, in the case where the SL protocol is selected;
3. the scheduling, i.e., the order in which each helper processes the offloaded tasks, in the case where the SL protocol is selected.

These decisions can be crucial for the training makespan by alleviating the effect of stragglers (i.e., the slowest devices) while fully utilizing the available resources. Hence, we formulate the problem of jointly making these decisions to minimize the makespan. An overview of the workflow of the HSFL system and the role of the optimization steps is depicted in the figure below (Figure 47). We also note that the considered framework can be easily generalized in the context of decentralized systems where the helpers and the aggregator may be devices/clients that are more powerful than others and with high connectivity.

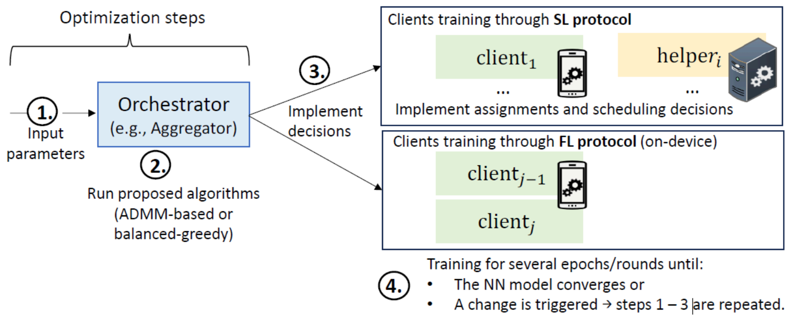

Optimization steps
1.
Input parameters
Orchestrator (e.g., Aggregator)
2.
Run proposed algorithms (ADMM-based or balanced-greedy)
3.
Implement decisions
Clients training through SL protocol
client_1
helper_i
...
Implement assignments and scheduling decisions
Clients training through FL protocol (on-device)
client_{j-1}
client_j
4.
Training for several epochs/rounds until:
The NN model converges or
A change is triggered → steps 1 – 3 are repeated.

Figure 47*: An overview of the workflow of the HSFL system, from profiling of processing and transmission times to the optimization of the workflow, and the system implementation.*

We analyse this problem and its challenges both theoretically (by proving it is NP-hard) and experimentally (using measurements from a realistic testbed). Therefore, we propose a solution method based on an intuitive decomposition of the problem into two subproblems, leveraging its inherent symmetry. The first one involves the training protocol selection for each client, the assignment, and the forward-propagation scheduling variables. The second one involves the backward-propagation scheduling variables. For the former, the Alternating Direction Method of Multipliers (ADMM) [76] is employed, while for the latter, a polynomial-time algorithm is provided (that is based on a known scheduling algorithm [77]). The advantage of employing ADMM lies in its versatility, allowing us to use techniques that may constrain the problem's solution space or tune its penalty parameters and stopping criteria, and thus, we tailor it to leverage the nature of the subproblem at hand.

Moreover, we propose a second solution method based on load balancing, that is more scalable, and thus, ideal for large problem instances. Specifically, we propose balanced-greedy that can be run by the orchestrator (e.g., the aggregator) and consists of three steps; it first decides on the training protocol per client (i.e., FL or SL), then on the client-helper assignments (for the clients for which the SL protocol was selected), and finally, on the scheduling at the helpers. We note that the time complexity of balanced-greedy[2] is smaller than the one of the ADMM-based method, which makes balanced-greedy ideal for large scale scenarios.

Finally, our numerical evaluations provide insights into the performance of the proposed methods, as well as the achieved gains in makespan in representative scenarios. We use CIFAR-10 [18] and two NN models: (i) ResNet101 [78], and (ii) VGG19 [79] for our training tasks. They are both deep convolutional NNs with 0.42 and 2.4 million parameters and are organized in 37 and 25 layers respectively. The testbed's devices include a RPi 3, a RPi 4, a Jetson GPU, a laptop, and a Virtual Machine. In our simulations, the values of the input parameters are set according to the profiling data of the testbed (for the computation times) and findings on Internet connectivity in France [80] (for the bandwidths of the network links). We explore two scenarios that represent two levels of heterogeneity in terms of devices, resources, and cut layers: Scenario 1 represents a system of low heterogeneity, while Scenario 2 a system of high heterogeneity.

Table 5 shows the relative gain in makespan in the HFSL setting vs. clients training only through SL. In detail, we consider a large scenario (row "large partition") according to the characteristics defined above and a small scenario (small partition) that consists of a set of 10 clients and 2 helpers (whose processing times follow the profiled times for VMs) and, initially, the profiled data of RPi 4 are used for the clients, while the bandwidth of all network connections belongs to the fastest class of the measurements of [80]. As we proceed, we alter the computing characteristics for a portion of the clients by slowing them down and similarly changing the bandwidth from the fastest class to the slowest one. The first row of Table 5 shows the relative gain in makespan in the HFSL setting for the corresponding experiments for the small partition. At first glance, we notice that when there are slow clients but good connections, having on-device training will not alter the makespan, except in the case where there are no slow connections and no slow devices, i.e., (0%, 0%), in which there is a slight acceleration of 4.8%. However, as more communication links get slower it is preferable to use on-device training (i.e., FL) even if the client's device is slower. We conclude that HFSL can decrease the makespan by up to 59% in the presence of slow network connections and devices that can process part-2 fast. This happens because model part-2 is small and the processing time on the devices is not significantly different from the processing time on the helper. As a result, the communication delay may be larger than the on-device computation delay. When having a larger part-2 it is more beneficial to offload, because, the on-device processing time becomes larger, while the offloading delay gets smaller. This is shown in the second row in Table 5, where the acceleration of HFSL is

[2] Balanced-greedy runs in O(JI) time, where J is the number of clients and I the number of helpers.

smaller compared to the previous case because fewer clients perform FL. Whereas, when there are faster clients with slow communication links we can see the effects of FL, where the makespan is decreased by up to 5.6%. In general, these findings demonstrate the importance of having a balance between the SL and FL, which is achieved through HFSL.

Table 5: Relative gain in makespan in the HFSL setting vs. clients training only through SL.

% of slow clients	100			75			20			0		
% of slow communication	0	50	100	0	50	100	0	50	100	0	50	100
small partition	0%	41.1%	42.3%	0%	20%	42%	0%	22.1%	41.1%	4.8%	57.8%	59%
large partition	0%	0%	0%	0%	1.17%	0%	0%	1.17%	0%	0%	1.17%	5.6%

Further, Table 6 shows the suboptimality and speedup achieved by the ADMM-based method when compared to Gurobi [81] that optimally solves the formulated optimization problem. The table shows the effectiveness of the ADMM-based method since it finds the optimal solution in most scenarios. There are some corner cases (e.g., 9.4%, 15.9%), but even then, there is a significant speedup compared to the solver, i.e., ×14, ×11.3, respectively. Finally, we can conclude that the proposed ADMM-based method finds the optimal solution in most problem instances and achieves up to x15.7 speedup compared to an ILP solver.

Table 6: Suboptimality and speedup achieved by the ADMM-based method compared to an ILP solver for HFSL for different problem instances (J denotes the number of clients, and I denotes the number of helpers).

		J	I	**suboptimality (%)**	**speedup (×)**
Scenario1	Resnet101	10	2	0	14.51
			5	9.4	14
		15	5	15.9	11.3
	VGG19	10	2	0	12.8
			5	0	13
		15	5	0	15.7
Scenario2	Resnet101	10	2	2.7	4
			5	0	4
		15	5	0	2.6
	VGG19	10	2	0	6.1
			5	0	10.5
		15	5	1.8	5.3

Finally, in Figure 48, we perform a sensitivity analysis with respect to the number of helpers in Scenario 1 where we depict the relative gains in makespan. Given the scenario's type and size, we employ balanced-greedy. We observe that, in a scenario of 100 clients and 1 helper, adding one more helper can dramatically decrease the batch makespan by up to 43.4%. Whereas, in the presence of 10 helpers, the relative gains of adding more helpers are decreasing. Such observations provide useful insights for real-life implementation of HFSL.

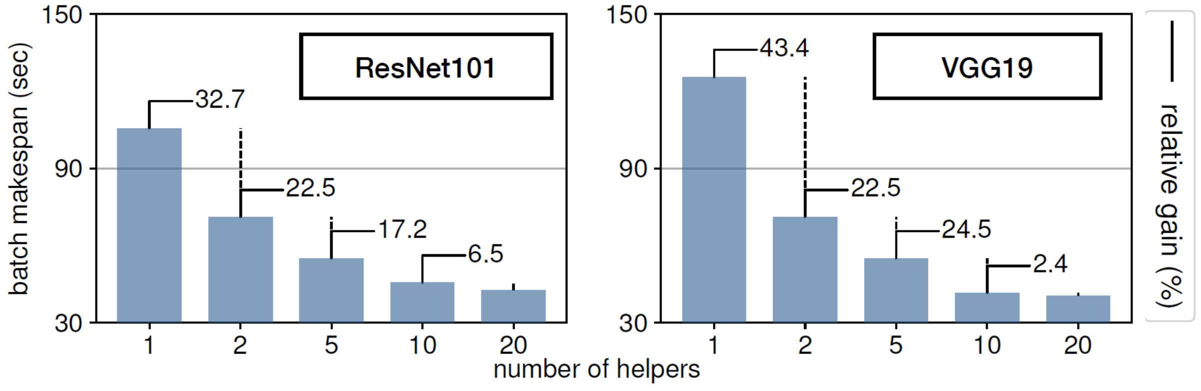


Figure 48: Makespan obtained by the balanced-greedy solution method in Scenario 1 (low heterogeneity) for J = 100 clients and varying number of helpers I.

7 Discussion and Future work

In this section, we discuss the contributions that the work package and its tasks have already made regarding the overall TaRDIS objectives per Grant Agreement (GA), and regarding the specific WP5 objectives per GA, and also identify the aspects that will be addressed during the last year of the project.

The TaRDIS project objectives are defined as follows (for more details, see the GA):

- *Objective 1*: Novel programming model for heterogeneous swarms: The aim is to create a language-independent, event-driven programming model that offers distribution abstractions and decentralised machine learning primitives.
- *Objective 2*: Development environment for correct-by-design heterogeneous swarms: The aim is to build a development environment with embedded semantic analyses to achieve a correctness-by-design approach.
- *Objective 3*: Decentralised intelligence for heterogeneous swarms: The aim is to provide support for decentralised intelligence for the purposes of heterogeneous swarms.
- *Objective 4*: Runtime support for distributed heterogeneous swarms: The aim is to provide decentralised algorithms and protocols for supporting the TaRDIS programming model at runtime.
- *Objective 5*: Interoperable execution environment: The aim is to ensure a high level of interoperability of TaRDIS distribution runtime, by supporting different devices and programming languages by formally specifying the protocols developed by the consortium.

The WP5 - Decentralised Machine learning, defines 3 main objectives (for more details see the GA), that correspond to different WP5 tasks:

- Develop a framework supporting decentralised learning and inference through AI/ML programming primitives (Task 5.1)
- Exploit and specialise the preceding for the planning deployment and orchestration of the complete TaRDIS framework through reinforcement learning and other relevant methodologies (Task 5.2)
- Develop novel lightweight ML techniques to enable decentralised and swarm learning in resource-constrained devices (Task 5.3)

7.1 Contributions to TaRDIS project objectives

WP5 contributes to 3 TaRDIS project objective (to specific results listed below):

- *Objective 1*: Novel programming model for heterogeneous swarms, linked with the following specific project result:
    - *R1.2 APIs for distribution, data management, and AI/ML (T5.1)*: the relevant tools for this result are PTB-FLA and MPT-FLA APIs (T-WP5-04) for the development of FL algorithms. The tools are already developed and available, which make this result addressed. However, possible future advances make the contributions to this result continuing.

- *Objective 3*: Decentralised intelligence for heterogeneous swarms, that addresses the following specific project results:
  - *R3.1 Techniques, algorithms, and models to support swarm intelligence. (T5.1, T5.2, T5.3):* The relevant tools for this result are the Flower-based FL tool (T-WP5-01, T-WP5-02, T-WP5-03), FAuNO (T-WP5-05) with PeersimGym, Fedra (T-WP5-09) and Lightweight ML tools (T-WP5-06, T-WP5-07, T-WP5-08). These tools are available and evolving, which make this result addressed, but still expanding.
  - *R3.2 Open-source implementation of decentralised algorithms for FL and Supervised Learning (SL). (T5.1, T5.3):* The relevant tools for this result are the Flower-based FL tool (T-WP5-01, T-WP5-02, T-WP5-03), Fedra (T-WP5-09) and Lightweight ML tools (T-WP5-06, T-WP5-07, T-WP5-08). These tools have been developed and are evolving in covering new decentralized algorithms. Therefore, this result is being addressed in continuation.
  - *R3.3 Contextual Machine Learning Operations (MLOps) solutions for swarm intelligence. (T5.1):* The relevant tool for this result is Flower-based FL model training. The contribution to this result is in progress. As already defined in Section 2, the Flower-based FL tool strives to simplify and automate various stages in the machine learning pipeline. This will include preparation and preprocessing techniques (T-WP5-02) and additional possibilities for inference and evaluation (T-WP5-03), beside the support for various training algorithms (T-WP5-01).
  - *R3.4 Dynamic peer-to-peer resource orchestration for the computing continuum. (T5.2):* The relevant tool for this result is FAuNO (T-WP5-05) with PeersimGym. The tools have been developed as described in Section 3, which addresses this result.
- *Objective 5:* Interoperable execution environment, that is linked with the following specific project result:
  - *R5.1 Open and extensible development environment supporting the TaRDIS' methodology and toolbox (T5.1):* The relevant tools for this result are PTB-FLA & MPT-FLA (T-WP5-04) as development environments for FL algorithm, which have been developed, as described in Section 2, making this result addressing continuously.

We now discuss these contributions in more detail, for specific TaRDIS WP5 tools below.

7.1.1 Flower-based FL tool contributions to the TaRDIS project objectives

The Flower-based FL tool (T-WP5-01, T-WP5-02, T-WP5-03) contributed to *Objective 3 of GA*: Decentralised intelligence for heterogeneous swarms, to the following results:

- R3.1 – Techniques, algorithms, and models to support swarm intelligence.
- R3.2 – Open-source implementation of decentralised algorithms for FL and Supervised Learning.
- R3.3 – Contextual MLOps solutions for swarm intelligence.

The Flower-based FL tool (T-WP5-01, T-WP5-02, T-WP5-03) is an integral part of the TaRDIS toolbox architecture (part of federated ML training unit in Figure 35), as an important aspect of the development of a swarm application. It provides the developer the possibility of utilising federated ML training strategies. The federated ML approaches within the TaRDIS architecture enable training on data sets that are distributed in diverse locations. The tool supports and guides the developer through the process of initiating the training. The developer needs to specify the target data set and choose an approach for training. This means that the use of the tool does not require expertise in the field. The tool also provides appropriate feedback when the training is finished, in terms of loss and accuracy. It supports model selection, parameter tuning and custom initialization, where a completely new or pre-existing model can be used for the training. It covers a set of FL algorithms, discussed in D5.1 [1] and in Section 2. Our plan for the future is to widen further the set of FL algorithms according to the needs, as well as to connect a set of data preparation and preprocessing, as well as inference and evaluation facilities to the tool.

7.1.2 PTB-FLA and MPT-FLA contributions to the TaRDIS project objectives

The work done on PTB-FLA and MPT-FLA (T-WP5-04) has contributed to the TaRDIS project objectives in the following way.

- O.1 Novel programming model for heterogeneous swarms
    - Contribution to the result R1.2. APIs for distribution, data management, and AI/ML

- O.3 Decentralised intelligence for heterogeneous swarms
    - Contribution to the result R3.2. Open-source implementation of decentralised algorithms for FL and SL
    - Contribution to the result R3.3. Contextual MLOPs solutions for swarm intelligence

- O.5 Decentralised intelligence for heterogeneous swarms
    - Contribution to the result R5.1. Open and extensible development environment supporting the TaRDIS' methodology and toolbox

The Python Testbed for Federated Learning Algorithms (PTB-FLA, T-WP5-04) was developed as a runtime environment for FL algorithms under development. Intentionally written in pure Python, it allows the algorithm designers to develop, execute and test their FL algorithms in an environment which is easy to install, because it has no external dependencies, and easy to fit to a small IoT edge device. The PTB-FLA supports the execution of centralised and decentralised federated learning algorithms, as well as the peer-to-peer data exchange used in Time Division Multiplexing (TDM) communication e.g., used for Orbit Determination and Time Synchronisation (ODTS) in Low Earth Orbit (LEO) satellite constellations. The PTB-FLA was explained in detail in D5.1.

MPT-FLA is a new framework that inherits all the advantages of PTB-FLA while overcoming its main limitation such that individual application instances may run on different network nodes like Personal Computers (PCs) and IoTs, primarily in edge systems. It is based on Python asynchronous Input/Output I/O (asyncio) abstractions (including asyncio coroutines, streams, and events), and runs on MicroPython. It is therefore a great match for smart IoTs and devices in edge systems.

The two frameworks define APIs that allow developers to easily implement the server and client callback functions and provide a ready-to-use environment for their execution in either a centralised or a decentralised way.

The PtbFla API comprises the following four functions (for details see the references in the section on published results below):

1. None PtbFla(noNodes, nodeId, flSrvId=0)
2. ret fl_centralized(sfun, cfun, ldata, noIters=1)
3. ret fl_decentralized(sfun, cfun, ldata, noIters=1)
4. PtbFla destructor

The MPT-FLA API comprises the following five members (the first is a function, whereas the next four are coroutines declared as such by the keyword async):

1. None PtbFla(noNodes, nodeId, flSrvId=0, mrIpAdr='localhost')
2. None async start()
3. ret async fl_centralized(sfun, cfun, ldata, pdata, noIters=1)
4. ret async fl_decentralized(sfun, cfun, ldata, pdata, noIters=1)
5. obs async get1Meas(peerId, odata).

Within the PTB-FLA and MPT-FLA we implemented the following examples:

- Federated map
- Centralised data averaging
- Decentralised data averaging
- Centralised logistic regression
- Decentralised logistic regression
- Centralised MNIST NN training and inference

The PTB-FLA and MPT-FLA development environments provide the solution for developing, testing and verifying the FL algorithms targeting the heterogeneous swarms. It is open-source, available on GitHub [35], and extensible allowing developers to extend its functionality if needed and develop their own examples.

The main task for the future work after D5.2 is to make necessary PTB-FLA adaptations and extensions to enable usage of PTB-FLA in the GMV use case. This task needs to be coordinated by NOVA and will be conducted by UNS in cooperation with NOVA and GMV.

7.1.3 FAuNO and PeersimGym contributions to TaRDIS project objectives

The contribution of T5.2 to TaRDIS objective 3: Decentralised intelligence for heterogeneous swarms is twofold:

- We are developing Reinforcement Learning-based (RL-based) orchestration of the TaRDIS runtime. The centralised version is completed. A federated RL framework is now in progress.
- It also provides decentralised (p2p) ML for Orbit Determination in satellite swarms. The centralised version is completed. A decentralised version is being developed.

The specific Project Results linked to this Objective are the following:

- R3.1 – Techniques, algorithms, and models to support swarm intelligence.
- R3.4 – Dynamic peer-to-peer resource orchestration for the computing continuum.

To summarize, the targeted results are partially met by the developed centralized implementations. The main focus for the future is to complete the development of the decentralized versions.

7.1.4 Fedra and lightweight ML tools contributions to TaRDIS project objectives

Objective 3 of GA is reached by promoting decentralised intelligence in swarm systems based on a decentralised FL framework (Fedra, T-WP5-09) that enables collaborative peer-to-peer intelligence sharing. The Fedra framework can support various ML algorithms and models being deployed directly at edge nodes, while keeping data and computation at a local level, promoting data privacy. Moreover, the work performed in T5.3 targets to offer lightweight ML inference techniques (T-WP5-06, T-WP-07, T-WP5-08) designed for nodes with limited computational capabilities. Finally, in the framework of WP5, a decentralised ML application was designed, developed and deployed for the EDP use case (smart home energy dynamics) in the TaRDIS environment. The ML-assisted application is based on RL, as well as assisted by forecasting ML models.

The specific Project Results here linked to this Objective are:

- R3.1 – Techniques, algorithms, and models to support swarm intelligence.
- R3.2 – Open-source implementation of decentralised algorithms for FL and Supervised Learning.

During the upcoming period of the TaRDIS project, the following activities will be conducted and finalized: (i) training of RL algorithms in the Fedra framework, specifically adapted for the energy use case; (ii) testing of the three lightweight ML inference techniques with real datasets and comparison of the resulting trade-off (accuracy vs computational resources); (iii) deployment of the early-exit models in real swarm nodes using Dexit tool; (iv) enhancement of the DRL model in a multi-agent framework, permitting the interaction between smart homes.

7.2 Contributions to WP5 objectives

The contributions regarding WP5 objectives can be categorised to 3 groups as follows:

1. *WP5 Objective 1:* Develop a framework supporting decentralised learning and inference through AI/ML programming primitives:

    a. An AI/ML library of implemented FL solutions for distributed AI applications was developed. It includes some widely applicable approaches, such as FL implementations of personalised and clustered federated learning (reported in D5.1), and distributionally robust FL (ongoing work). Also, the application of autoencoders and k-means for the process of anomaly detection for the ACT use case has been investigated. A tool for Flower-based FL (T-WP5-01/02/03), that supports the developer during the training setup, has been introduced. An approach for customised client-server message exchange in Flower implementations has been also considered.
    b. PTB-FLA & MPT-FLA (T-WP5-04) development environments for implementing the FL programming primitives were introduced.
    c. Precise ML Algorithms for Orbit Determination were proposed, which will be decentralised in the second half of the project.

   d. A decentralised framework (Fedra, T-WP5-09) was designed and developed in the first half of the project lifetime, as an additional framework that enables model-agnostic FL learning.

2. *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

   a. The Federated AI Network Orchestrator (FAuNO, T-WP5-05) has been developed, consisting of a system allowing RL agents under a Markov Game framework to solve the task offloading problem, supported by a training environment PeersimGym for the decentralised agents.

3. *WP5 Objective 3:* develop novel lightweight ML techniques to enable decentralised and swarm learning in resource-constrained devices

   a. MPT-FLA has been developed. It represents a Micro-Python implementation of PTB-FLA (T-WP5-04) that allows the development of federated learning algorithms in resource-constrained devices, such as Raspberry Pi Pico W boards, Husarion ROSbot, etc.

   b. Communication efficient vertical FL via compressed error feedback is proposed

   c. Three Lightweight ML tools methods were introduced: early-exit (EE, T-WP5-06) of inference, knowledge distillation (KD, T-WP5-07) and pruning (T-WP5-07) techniques.

We now discuss these contributions in more detail, for specific TaRDIS WP5 tools below.

7.2.1 Flower-based FL tool contributions to the TaRDIS WP5 objectives

The Flower-based FL tool (T-WP5-01, T-WP5-02, T-WP5-03) contributes to the following WP5 objective:

WP5 Objective 1: to develop a framework supporting decentralised learning and inference through AI/ML programming primitives.

The contribution has a few dimensions. First, it includes a list of widely applicable approaches, such as FL implementations of personalised and clustered FL (both reported in D5.1), and distributionally robust FL and anomaly detection (see Section 2). Also, the tool contributes to the ACT use case specific needs. The application of autoencoders and k-means for the process of anomaly detection for the ACT use case has been investigated (see Sections 2 and 6). Additionally, the developed Flower-based FL tool supports the developer during the training setup and makes the process of starting the training straightforward. Finally, we also made an implementation-wise progress, by implementing a custom client-server message exchange approach in the Flower implementation of the DR FL. The tool will evolve further in the upcoming period, as described above.

7.2.2 PTB-FLA and MPT-FLA contributions to the TaRDIS WP5 objectives

The work done on PTB-FLA and MPT-FLA (T-WP5-04) contributed to the following WP5 objectives.

WP5 Objective 1: to develop a framework supporting decentralised learning and inference through AI/ML programming primitives.

WP5 Objective 3: to develop novel light-weight ML techniques to enable decentralised and swarm learning in resource-constrained devices.

Regarding O.1, as already mentioned above, both PTB-FLA and MPT-FLA are the FL frameworks supporting decentralised learning and inference through AI/ML programming primitives.

Regarding O.3, the MPT-FLA framework was experimentally validated on the Wireless Fidelity (WiFi) network, consisting of one WiFi router Belkin F5D7234-4, two Raspberry Pi Pico W boards, and one PC, by using the four adapted algorithm examples originally developed for the PTB-FLA framework. The MPT-FLA successfully passed this experimental validation because, as expected, the adapted algorithms produced the same numerical results as the originals, and this was the sole goal of this experiment validation. The validation showed its applicability in resource-constrained devices.

7.2.3 Fedra contribution to the TaRDIS WP5 objectives

The Fedra framework (T-WP5-09) contributes to the following objective:

WP5 Objective 1: to develop a framework supporting decentralised learning and inference through AI/ML programming primitives

Fedra was designed and developed in the first half of the project lifetime, as an additional framework that enables model-agnostic FL learning.

7.2.4 Orbit determination ML algorithms contributions to the TaRDIS WP5 objectives

The contribution is related to the following WP5 objective:

WP5 Objective 1: to develop a framework supporting decentralised learning and inference through AI/ML programming primitives

Precise ML Algorithms for Orbit Determination are planned to be decentralised in the second half of the project.

7.2.5 FAuNO and PeersimGym contributions to the TaRDIS WP5 objectives

The tools FAuNO (T-WP5-05) and PeersimGym developed under T5.2 contribute to the following WP5 objective:

WP5 Objective 2: exploit and specialise the preceding for the planning, deployment, and orchestration of the complete TaRDIS framework through reinforcement learning and other relevant methodologies

The team is developing the Federated AI Network Orchestrator (FAuNO), consisting of a system allowing RL agents under a Markov Game framework to solve the task offloading problem. For this purpose, the team developed a training environment for the decentralised agents, PeersymGym.

7.2.6 Lightweight ML techniques contribution to the TaRDIS WP5 objectives

WP5 Objective 3: develop novel lightweight ML techniques to enable decentralised and swarm learning in resource-constrained devices

Three methods were investigated for this objective: early-exit (EE, T-WP5-06) of inference, knowledge distillation (KD, T-WP5-07) and pruning (T-WP5-08) techniques. Four tools were developed and will be incorporated in the TaRDIS toolbox to provide these functionalities.

7.2.7 Communication efficient vertical federated learning contribution to the TaRDIS WP5 objectives

WP5 Objective 3: develop novel lightweight ML techniques to enable decentralised and swarm learning in resource-constrained devices

Communication efficient vertical FL via compressed error feedback is being developed.

8 Contribution to TaRDIS KPIs

We present the KPIs that are related to decentralised ML specific tools. In D2.2, an analysis of the overall requirements was performed, where the first version of the requirements list was identified (please see D2.2 for the complete list of KPIs). The table below (Table 7) lists the relevant KPIs and the decentralised ML TaRDIS tools that are related to them. Note that O means Objective, and B means Baseline, in KPIs IDs, as denoted in D2.2. We present a few preliminary results regarding the contributions to KPIs. However, these will be aligned with the descriptions of the KPIs from D7.2 in the future. In D7.2, a preliminary evaluation of the TaRDIS toolbox was carried out, where the appropriate measurement methodologies and related requirements were identified for the KPIs. We discuss the KPIs listed in Table 7, per individual tools below.

Table 7: KPIs relevant for decentralised ML specific tools.

ID	Description	WP5 tools
K-O-1.3	Decrease median development time by 25%	PTB-FLA based model training
K-O-3.1	Use TaRDIS ML to autonomously manage system operations	Federated AI Network Orchestrator (FAuNO)
K-O-3.2	Improved edge orchestration	Federated AI Network Orchestrator (FAuNO)
K-O-3.3	Reduced Transmission overhead by 20%	Flower-based FL model training Fedra and the lightweight ML tools
K-O-3.4	Model reduction/compression increased by 15%	Fedra and the lightweight ML tools
K-O-3.5	Reduced model training time by 25%	Fedra and the lightweight ML tools
K-B-07	FL training latency	Flower-based FL model training Fedra and the lightweight ML tools
K-B-08	FL storage/RAM requirements per node	Flower-based FL model training Fedra and the lightweight ML tools
K-B-10	FL accuracy	Flower-based FL model training Fedra and the lightweight ML tools

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

As shown in Table 7, this tool is related to the following KPIs:

- K-B-07: FL training latency
- K-B-10: FL accuracy
- K-B-08: FL storage/Random-Access Memory (RAM) requirements per node
- K-O-3.3: Reduced transmission overhead by 20% (w.r.t. FedAvg)

The tool addresses the KPI K-B-07: FL training latency, in two ways. The first approach is to measure the runtimes by executing a FL training algorithm with different numbers of participating client nodes. These simulations have been carried out on a cluster environment containing 16 nodes, 8 Intel i7 5820k 3.3GHz and 8 Intel i7 8700 3.2GHz CPU - 96 cores and 16GB DDR4 RAM/node, interconnected by a 10 Gbps network. Although a cluster environment is not a natural choice for federated learning, it may serve as an excellent setup for evaluations and running large-scale simulations.

The results are shown for the pFedMe algorithm (reported in D5.1), in Figure 49. The simulations were launched on CPUs with a different number of Flower clients *c=[2,3,..,10]*, where each client represents a separate cluster node. The experiments were performed on the Fashion MNIST data set, containing 60000 training examples. It can be observed that the algorithm scales well, reaching a 'sweet spot' at 6 clients, where the execution time has the lowest value. The plot also shows that the differences in timings are not drastic up to 7 clients. After that point, the time starts to grow while increasing the number of clients, which is a typical behavior of a parallel program. At that point, the gains of parallelisation for this data set are becoming lower than the expenses of synchronising.

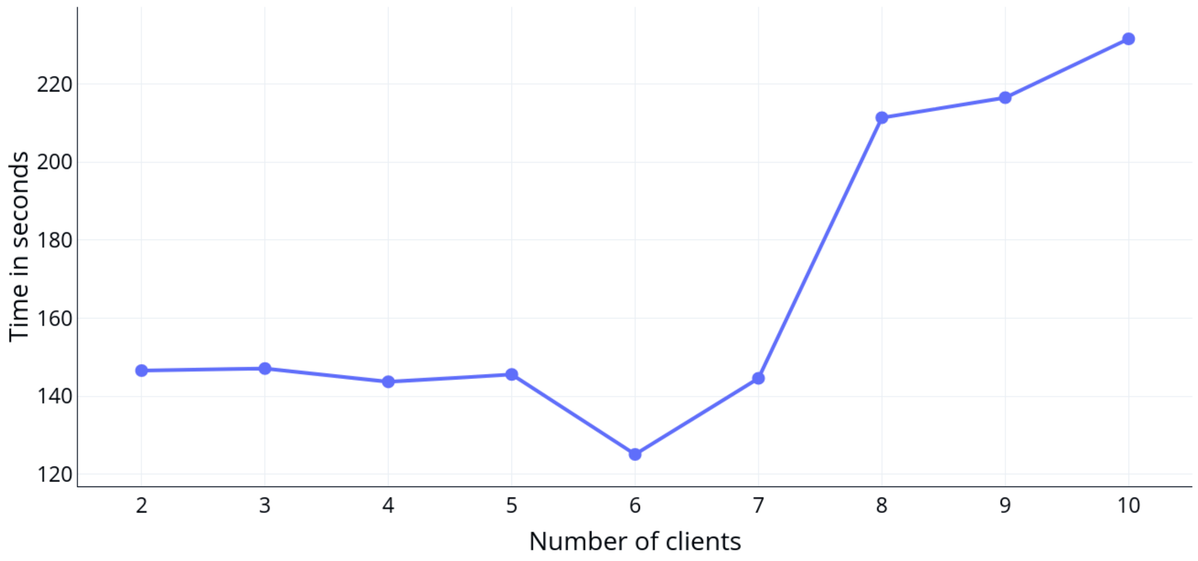


Figure 49*: Scaling properties of the pFedMe implementation on a cluster.*

The second set of experiments that contributes to K-B-07 is the measure of the scalability of a serial implementation of the Autoencoder, used for the ACT use case. See Figure 38 in Section 6 above. It can be observed that the increase of training time with respect to increased number of data points is almost linear. This is a preliminary result that seems promising, as it can be expected that the parallel, federated version of the algorithm will scale well, similarly as the pFed me algorithm, in Figure 49.

Regarding the KPI K-B-10: FL accuracy, the tool already provided measured accuracy values for the developed pFedMe algorithm (reported in D5.1). Please see D5.1 for additional details. Also, we measured the accuracy values for the autoencoder for the ACT use case. The results seem promising, giving accuracies of 0.94 and 0.82 (versus ground truth). For more details on the results, see Section 6.

The KPI K-B-08: FL storage/RAM requirements per node, will be addressed also by running a set of experiments on a cluster environment. During FL model training, the storage and RAM resources of the participating nodes will be measured.

For K-O-3.3: Reduced transmission overhead by 20% (wrt FedAvg) will be measured regarding the network load of the model weights exchange during the FL process.

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

The work done on PTB-FLA and MPT-FLA has contributed to the following TaRDIS KPI.

- K-O-1.3 Decrease median development time by 25%.

In [28] we reported on the adapted development paradigms performance in terms of human labour (in working hours) and size of ChatGPT context (in number of characters with spaces) needed to develop the logistic regression PTB-FLA code. We would like to emphasise that data on human labour should be treated as rough estimates because it is based on our freeform notes in private diaries. Data on ChatGPT context size is exact and we got them by the text editor.

In the following table (Table 8), the left part relates to human labour and the right part relates to the ChatGPT context size. The top part contains raw (input) data, and the bottom part contains calculated (output) data; some of the fields are not applicable (but this should be obvious, so we skip explaining these exceptions). There are two types of calculated data: (i) the working speed-up that is defined as the ratio of working hours, and (ii) the ChatGPT context size reduction that is defined as the ratio of ChatGPT context sizes.

Table 8: Adapted paradigms performance data.

	Human labour [h]			Context size [ch with spaces]	
Phase	4-Ph human	4-Ph GPT	2-Ph GPT	4-Ph GPT	2-Ph GPT
Phase 2	8	4	4	2462	2685
Phase 3	12	4		2593	
Phase 4	4	4		2334	
Total	24	12	4	7389	2685
	Speed up:	**2**	**6**	**Reduction:**	**2.75**

The calculated data in the bottom raw reveal encouraging results, the adapted 4-phases development paradigm for ChatGPT achieved the speed up of 2 times over the original 4-phases development paradigm for humans, whereas the adapted 2-phase development paradigm for ChatGPT achieved the speed up of 6 times over the original 4-phase development paradigm for humans. However, one should keep in mind that these impressive results are based on rough estimates of working hours.

Regarding the ChatGPT context sizes, the calculated data in the bottom row of Table 8 reveal that the adapted 2-phases development paradigm for ChatGPT achieves the context size reduction of 2.75 times over the adapted 4-phases development paradigm for ChatGPT i.e., it makes using ChatGPT 2.75 times cheaper.

8.3 KPIs for the FAuNO tool (T-WP5-05)

The following KPIs can be demonstrated for the tool FAuNO, that can be directly measured and compared to baseline results:

- K-O-3.1 Use TaRDIS ML to autonomously manage system operations (used by 50% of use cases). FAuNO is being developed as a generic AI orchestration tool and can be applied to the GMV use case, by specialisation and reconfiguration.
- K-O-3.2 Improved edge orchestration (15% faster response time, 20% faster event processing throughput). When compared to standard heuristic algorithms, like Least Queues, vanilla RL agents already show improvement. We expect that FRL and specially designed agents will attain the KPI.

8.4 KPIs for the Fedra framework (T-WP5-09) and Lightweight ML tools (T-WP5-06/07/08)

Fedra and the lightweight ML tools are linked with the following KPIs that can be directly measured and compared to baseline results:

- K-B-07: FL training latency, measured by executing the FL training with SL and RL algorithm and varying number of participating nodes
- K-B-08: FL storage/RAM requirements per node, obtained by executing the FL training in a practical implementation scheme (optionally with virtual machines representing decentralised FL nodes) and measuring the storage and RAM resources of the participating nodes. Moreover, the storage/RAM requirements and inference latency can be also measured when the EE, KD or pruning methods are employed for the model hosted at the edge nodes.
- K-B-10: FL accuracy, by obtaining the testing error of the trained models and quantifying the validation/inference error of the lightweight models.
- K-O-3.3: Reduced transmission overhead, measuring the network load of the model weights exchanged during the FL process, as well as the data exchange between the nodes during the inference in the case of EE.
- K-O-3.4: Model reduction/compression, measuring the compression rate of the neural network when using the pruning method.
- K-O-3.5: Reduced model training time by 25%, measured by executing FL training in the Fedra framework.
- Develop at least 3 different lightweight techniques (KD, EE, pruning) and demonstrate SL and RL algorithms in the TaRDIS use cases.

It should be noted that several of the KPIs have been partially measured according to the measurement methodology presented in D7.2 - Report on the preliminary evaluation of the TaRDIS toolbox. These KPIs will be presented in the evaluation of the Tardis toolbox results for the associated use cases, as well as in the final WP5 deliverable D5.3.

Funded by the European Union

9 Conclusion

This document has reported the advances and ongoing work on the tasks regarding the development of decentralized machine learning approaches. It contains the contributions for the period after D5.1 submission. It has presented the current statuses of frameworks that support AI/ML primitives, AI-driven planning, deployment and orchestration, and lightweight, energy efficient ML techniques. We also described the positioning of these approaches in TaRDIS. Additionally, we listed the relevant KPIs and objectives, supported by some results and plans for addressing them in the future. We also described the novelties and current state of the art regarding the ML modelling of all TaRDIS use cases.

The TaRDIS project tasks T5.1, T5.2 and T5.3 will continue with their activities, by working on improving and expanding their solutions, in order to support the development of the TaRDIS toolbox, while constantly focusing on the needs of the use cases. The results of these activities will be documented in the next (and final) deliverable for the work package (D5.3), which is due on M34.

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] A. Armacki, H. Sharma, D. Bajović, D. Jakovetić, M. Chakraborty, S. Kar. Distributed Gradient Clustering: Convergence and the Effect of Initialization. Asilomar conference on signals, systems and computers, October 27-30, 2024

[3] D. Bajović, D. Jakovetić, S. Kar, M. Vuković. Tackling heavy-tailed noise in distributed estimation: Asymptotic performance and tradeoffs. Telecommunications Forum, TELFOR, November 26-27, 2024

[4] 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

[5] 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

[6] 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

[7] Issaid, C.B., Elgabli, A., Bennis, M. (2022). DR-DSGD: A Distributionally Robust Decentralized Learning Algorithm over Graphs. *Transaction on Machine Learning Research, 2022.* https://openreview.net/pdf?id=VcXNAr5Rur

[8] Zecchin, M., Kountouris, M., Gesbert, D. (2022). Communication-Efficient Distributionally Robust Decentralized Learning. *Transaction on Machine Learning Research, 2023.* https://arxiv.org/abs/2205.15614

[9] Mohri, M., Sivek, G., Suresh, A.T.. (2019). Agnostic Federated Learning. Proceedings of the 36th International Conference on Machine Learning, in Proceedings of Machine Learning Research 97:4615-4625, Available from https://proceedings.mlr.press/v97/mohri19a.html.

[10] Ester, M., Kriegel H.-P., Sander, J., Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96). AAAI Press, 226–231.

[11] Flower documentation, retrieved December 2, 2024, https://flower.ai/docs/framework/tutorial-series-customize-the-client-pytorch.html

[12] 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

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, DG Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, USA, OSDI’16, pp 265–283, 2016, https://doi.org/10.48550/arXiv.1605.08695

[14] D.P. Kingma, J. B. Adam: A method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, https://doi.org/10.48550/arXiv.1412.6980

[15] S. Lloyd, Least squares quantization in PCM. IEEE Transactions on Information Theory 28 (2): 129–137, 1982, DOI: 10.1109/TIT.1982.1056489

[16] J. MacQueen, Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, Berkeley, Calif., pp 281–297, 1967

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P.

Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830, 2011, https://doi.org/10.48550/arXiv.1201.0490

[18] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009, retrieved November 20, 2024, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[19] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142. doi: 10.1109/

[20] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).*

[21] Gao Y., Kim, M., Thapa, C., Abuadbba, A., Zhang, Z., Camptepe, S. Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things. IEEE Transactions on Computers, vol. 71, no. 10, pp. 2538-2552, 1 Oct. 2022, doi: 10.1109/TC.2021.3135752

[22] ChatGPT helps humans creating federated learning apps on PTB-FLA, retrieved November 25, 2024,
https://www.project-tardis.eu/news/2024/06/11/chatgpt-helps-humans-creating-federated-learning-apps-on-ptb-fla/

[23] The PTB-FLA successor MPT-FLA advances to edge systems, retrieved November 20, 2024, https://www.project-tardis.eu/blog/2024/06/25/the-ptb-fla-successor-mpt-fla-advances-to-edge-systems/

[24] Validation of MPT-FLA, retrieved November 12, 2024,
https://www.youtube.com/watch?v=QQJ-xs7ZG3

[25] M. Popovic, M. Popovic, I. Kastelan, M. Djukic and S. Ghilezan, "A Simple Python Testbed for Federated Learning Algorithms," 2023 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, 2023, pp. 148-153, https://doi.org/10.1109/ZINC58345.2023.10173859

[26] Popovic, M., Popovic, M., Kastelan, I., Djukic, M., Basicevic, I. (2024). A Federated Learning Algorithms Development Paradigm. In: Kofroň, J., Margaria, T., Seceleanu, C. (eds) Engineering of Computer-Based Systems. ECBS 2023. Lecture Notes in Computer Science, vol 14390. Springer, Cham. https://doi.org/10.1007/978-3-031-49252-5_4

[27] Prokić, I., Ghilezan, S., Kašterović, S., Popovic, M., Popovic, M., Kaštelan, I. (2024). Correct Orchestration of Federated Learning Generic Algorithms: Formalisation and Verification in CSP. In: Kofroň, J., Margaria, T., Seceleanu, C. (eds) Engineering of Computer-Based Systems. ECBS 2023. Lecture Notes in Computer Science, vol 14390. Springer, Cham. https://doi.org/10.1007/978-3-031-49252-5_25

[28] M. Popovic, M. Popovic, I. Kastelan, M. Djukic and I. Basicevic, "Developing Elementary Federated Learning Algorithms Leveraging the ChatGPT," 2023 31st Telecommunications Forum (TELFOR), Belgrade, Serbia, 2023, pp. 1-4,
https://doi.org/10.1109/TELFOR59449.2023.10372714

[29] M. Popovic, M. Popovic, I. Kastelan, M. Djukic and I. Basicevic. PTB-FLA Development Paradigm Adaptation for ChatGPT. Computer Science and Information Systems, vol. 21, no. 4, pp. 1-25, 2024. https://doi.org/10.2298/CSIS231224036P

[30] M. Popovic, M. Popovic, I. Kastelan, M. Djukic and I. Basicevic, "MicroPython Testbed for Federated Learning Algorithms" preprint on arXiv: https://arxiv.org/abs/2405.09423

[31] Popovic, M., Popovic, M., Djukic, M., Basicevic, I. Towards Formal Verification of Federated Learning Orchestration Protocols on Satellites. arXiv:2410.13429 [cs.DC], 2024.
https://doi.org/10.48550/arXiv.2410.13429

[32] Popovic, M., Popovic, M., Kastelan, I., Djukic, M., Basicevic, I.: PTB-FLA Development Paradigm Adaptation for ChatGPT. Computer Science and Information Systems, Vol. 21, No. 4, 1269–1292. (2024), https://doi.org/10.2298/CSIS231224036P

[33] Pavle Vasiljević (2024). Distribuirani lanser za okruženje za federativno učenje MPT-FLA. Fakultet Tehničkih nauka, Novi Sad. https://www.rt-rk.uns.ac.rs/sites/default/files/bachelor_radCyr.pdf

[34] Distributed launcher for the MPT-FLA federated learning framework, retrieved December 2, 2024, https://github.com/LinguineP/distributedLauncher

[35] PTB-FLA, retrieved November 27, 2024, https://github.com/miroslav-popovic/ptbfla

[36] MNIST neural network from scratch, retrieved November 19, 2024, https://github.com/SohamP2812/MNIST-Neural-Network-from-Scratch/blob/main/MNIST_N

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

[38] Marko Nikolovski (2024). Distribuirane aplikacije za obučavanje neuralne mreže MNIST na okruženjima PTB-FLA i MPT-FLA. Fakultet Tehničkih Nauka, Novi Sad. https://www.rt-rk.uns.ac.rs/sites/default/files/Diplomsi-rad.pdf

[39] Sun, T., D. Li, and B. Wang. Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.4 (2022): 4289-4301. doi: 10.1109/TPAMI.2022.3196503

[40] F. Metelo, S. Racković, P. Á. Costa, C. Soares, PeersimGym: An Environment for Solving the Task Offloading Problem with Reinforcement Learning, Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track, ECML-PKDD 2024

[41] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Kimov, O.. Proximal Policy Optimization Algorithms. arXiv:1707.06347, arXiv, 28 Aug. 2017. arXiv.org, http://arxiv.org/abs/1707.06347.

[42] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, D. Huba. Federated Learning with Buffered Asynchronous Aggregation. arXiv:2106.06639, arXiv, 7 Mar. 2022. arXiv.org, http://arxiv.org/abs/2106.06639.

[43] McMahan, H.B., Moore, E., Ramage, D., Hampson, S., & Arcas, B.A. (2016). Communication-Efficient Learning of Deep Networks from Decentralized Data. *International Conference on Artificial Intelligence and Statistics*.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis. Human-Level Control through Deep Reinforcement Learning. Nature, vol. 518, no. 7540, Feb. 2015, pp. 529–33. DOI.org (Crossref), https://doi.org/10.1038/nature14236.

[45] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. Proceedings of The 33rd International Conference on Machine Learning, in Proceedings of Machine Learning Research 48:1928-1937 Available from https://proceedings.mlr.press/v48/mniha16.html.

[46] T. Rausch, C. Lachner, P. A. Frangoudis, P. Raith, S. Dustdar, TU Wien. Synthesizing plausible infrastructure configurations for evaluating edge computing systems. In: 3rd USENIX Workshop HotEdge 20. 2020, https://www.usenix.org/conference/hotedge20/presentation/rausch

[47] Tsinos, C., Spantideas, S., Giannopoulos, A., & Trakadas, P. (2023). Over-the-Air Computation with Quantized CSI and Discrete Power Control Levels. *Wireless Communications and Mobile Computing*, *2023*(1). https://doi.org/10.1155/2023/8559701

[48] Zetas, M., Spantideas, S., Giannopoulos, A., Nomikos, N., & Trakadas, P. (2024). Empowering 6G maritime communications with distributed intelligence and over-the-air model sharing. *Frontiers in Communications and Networks*, *4*, https://doi.org/10.3389/frcmn.2023.1280602

[49] Paralikas, I., Spantideas, S., Giannopoulos, A., & Trakadas, P. (2024, June). Lightweight Inference by Neural Network Pruning: Accuracy, Time and Comparison. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 248-257). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-63219-8_19

[50] M. Christopoulos, S. Spantideas, A. Giannopoulos, P. Trakadas. 2024. Deep Reinforcement Learning for Smart Home Temperature Comfort in IoT-Edge Computing Systems. In Proceedings of the 1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (MECC '24). Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3642975.3678961

[51] A. E. Giannopoulos, S. T. Spantideas, M. Zetas, N. Nomikos and P. Trakadas. FedShip: Federated Over-the-Air Learning for Communication-Efficient and Privacy-Aware Smart Shipping in 6G Communications. In *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 12, pp. 19873-19888, Dec. 2024, doi: 10.1109/TITS.2024.3468383

[52] The CIFAR-10 dataset, retrieved November 28, 2024, https://www.cs.toronto.edu/~kriz/cifar.html

[53] P. Valdeira, J. Xavier, C. Soares, Y. Chi. Communication-efficient Vertical Federated Learning via Compressed Error Feedback. European Signal Processing Conference, EUSIPCO 24, August 26-30, 2024. https://eurasip.org/Proceedings/Eusipco/Eusipco2024/pdfs/0001037.pdf

[54] So, J., Hsieh, K., Arzani, B., Noghabi, S., Avestimehr, S., & Chandra, R. (2022). Fedspace: An efficient federated learning framework at satellites and ground stations. arXiv preprint arXiv:2202.01267.

[55] 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

[56] 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

[57] 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

[58] 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

[59] Prokić, I., Ghilezan, S., Kašterović, S., Popovic, M., Popovic, M., Kaštelan, I. (2024). Correct Orchestration of Federated Learning Generic Algorithms: Formalisation and Verification in CSP. In: Kofroň, J., Margaria, T., Seceleanu, C. (eds) Engineering of Computer-Based Systems. ECBS 2023. Lecture Notes in Computer Science, vol 14390. Springer, Cham. 274-288. https://doi.org/10.1007/978-3-031-49252-5_25

[60] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, et al., "Engineering self-adaptive systems through feedback loops", Software engineering for self-adaptive systems, Lecture Notes in Computer Science, vol 5525. Springer, Berlin, Heidelberg. pp. 48-70, 2009. https://doi.org/10.1007/978-3-642-02161-9_3

[61] D7.2 - Report on preliminary validation of the toolbox, 2024, TaRDIS project

[62] 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

[63] 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

[64] N. Davari, B. Veloso, R.P. Ribeiro, P.M. Pereira, J. Gama: Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry. In: 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA). pp. 1–10. IEEE, 2021, DOI:10.1109/DSAA53316.2021.9564181

[65] M. Barros., B. Veloso, P.M. Pereira, R.P. Ribeiro, J. Gama: Failure detection of an air production unit in the operational context. In: IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning, pp. 61–74. Springer, 2020, https://doi.org/10.1007/978-3-030-66770-2_5

[66] W. Todo, B. Laurent, J. Loubes, M. Selmani: Dimension Reduction for time series with Variational AutoEncoders, 2022, https://doi.org/10.48550/arXiv.2204.11060

[67] J. Xu, H. Wu, J. Wang, M. Long, Anomaly Transformer: Time Series Anomaly Detection with

Association Discrepancy, International Conference on Learning Representations, ICLR 2022, https://doi.org/10.48550/arXiv.2110.02642

[68] F. Caldas and C. Soares. Precise and Efficient Orbit Prediction in LEO with Machine Learning using Exogenous Variables. *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, pp. 1-8, doi: 10.1109/CEC60901.2024.10611996.

[69] F. Caldas, C. Soares. Machine learning in orbit estimation: A survey. Acta Astronautica, Volume 220, 2024, Pages 97-107, ISSN 0094-5765, https://doi.org/10.1016/j.actaastro.2024.03.072.

[70] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, Volume 378, 2019, pp 686-707,ISSN 0021-9991,https://doi.org/10.1016/j.jcp.2018.10.045

[71] Izzo, D., Acciarini, G., Biscani, F. (2024). NeuralODEs for VLEO simulations: Introducing thermoNET for Thermosphere Modeling. 29th International Symposium on Space Flight Dynamics

[72] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In Conference on artificial intelligence and statistics, AISTATS. PMLR, 2017, pp. 1273–1282.

[73] J. Tirana, D. Tsigkari, G. Iosifidis, and D. Chatzopoulos, “Workflow optimization for parallel split learning,” in Proc. of IEEE International conference on computer communications, INFOCOM, 2024.

[74] Liu, X., Deng, Y. and Mahmoodi, T., 2022. Wireless distributed learning: A new hybrid split and federated learning approach. IEEE Transactions on Wireless Communications, 22(4), pp.2650-2665. doi: 10.1109/TWC.2022.3213411.

[75] Workflow optimization for parallel split learning, retrieved December 15, 2024, https://github.com/jtirana98/SFL-workflow-optimization

[76] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine learning, 2011, doi: 10.1561/2200000016

[77] K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. Operations Research, vol. 31, no. 2, pp. 381–386, 1983

[78] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proc. of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778

[79] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014

[80] D. Belson, “State of the Internet Q4 2016 report,” Akamai Technologies, vol. 9, no. 4, 2017.

[81] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023. [Online]. Available: https://www.gurobi.com

Funded by the European Union