MANOLO

CLOUD-EDGE EFFICIENT & TRUSTWORTHY AI

D4.1 Cloud-edge resources, infrastructure, and AI models mapping

TUBS

30/06/2025

Funded by the European Union

Document Information

Issued by:	**TUBS**
Issue date:	25/04/2025
Due date:	30/06/2025
Work package leader:	UPC
Dissemination level:	PU - Public

Document History

Version	Date	Modifications made by
0.1	25/04/2025	TUBS, ToC shared with all Partners
0.2	19/05/2025	FDI, draft of the chapter 4
0.3	27/05/2025	UPC, draft of the chapter 7
0.4	28/05/2025	TUBS, draft of the chapter 5
0.5	30/05/2025	TUBS, draft of the chapter 1
0.6	04/06/2025	ATOS, draft of the chapter 6
0.7	11/06/2025	UPC, TUBS, draft of the chapter 2
0.8	13/06/2025	TUBS, UPC, draft of the chapter 3
0.9	17/06/2025	TUBS, draft of the chapter 8
1.0	18/06/2025	First version of the document by TUBS
2.0	25/06/2025	Second version by TUBS after the quality review by Q-PLAN and FDI

Authors

Name(s)	Beneficiary
Mounir Bensalem , Fin Gentzen, Admela Jukan	TUBS
Muhammad Imran, Eva Marin Tordera	UPC
Giorgos Levis, Manolis Kelaidis	FDI
Francesco D'Andria	ATOS

In case you want any additional information or you want to consult with the authors of this document, please send your inquiries to: a.jukan@tu-bs.de

Quality Reviewers

Name(s)	Beneficiary
Georgios Spyridopoulos	Q-PLAN
Manolis Kelaidis	FDI

Executive Summary

This deliverable presents the progress of the MANOLO WP4 components, produced by the MANOLO WP4 and use case partners, to describe an architecture that can fulfil the project's objectives and use cases and document the technical requirement of each component.
This document also aims to align all Consortium partners on a clarified workflow to understand the intra-, and inter-components communication for all WP4 modules, as well as aligning the architecture with the use cases. This is particularly crucial as different partners work independently within various technical Work Packages throughout the system's implementation.
The WP4 architecture will evolve in response to feedback obtained during the development and validation phases. The development of each part of WP4 will be finalized and reported in D4.2 “Cloud-edge resources, infrastructure, and AI models mapping v.2” at project month 30.

Table of Contents

List of Figures

List of Tables

List of Terms and Definitions

Table 1. Terms and Definitions

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CEC	Cloud Edge Continuum
D4.1	Deliverable 4.1 – the first official report for Work Package 4, focused on cloud-edge resources infrastructure and AI models mapping
FL	Federated Learning
GDPR	General Data Protection Regulation
ML	Machine Learning
WP	Work Package – a defined section of a project plan that groups related tasks and objectives.
YAML	Yet Another Markup Language - a human-readable data-serialization format used for configuration files (e.g., in KOBE experiments).
Z-Inspection®	A methodology for assessing the trustworthiness of AI systems through ethical and socio-technical analysis.

1. Introduction

1.1 Scope of Deliverable

This document presents the outcomes of the collaborative effort performed as part of Work Package 4 in defining the AI Model Function Allocation across Cloud-edge Continuum. Specifically, it addresses the tasks of Federated Learning (T4.1), Cloud-edge Resource and Infrastructure Mapping (T4.2), Trustworthy-driven Policy Validation (T4.3), and Trustworthy & Infrastructure Oriented Model and Learning Allocation (T4.4).

This document outlines the core structural elements of the WP4 components. It details their architecture, functionalities, and interactions, capturing how these elements collectively address system requirements. Additionally, the development progress is presented, highlighting the steps taken to refine the design and implementation. Finally, this document demonstrates how the WP4 components integrate with relevant use cases, ensuring a seamless alignment with the project's overarching objectives.

1.2 Structure of the document

This document is structured into 9 main sections and supporting Appendices:

1. **Introduction**: This section explains the purpose of the D4.1 document within the context of the WP4 components. It offers an overview of their core structural elements, functionalities, and interactions, along with the progress in terms of architecture and development.

2. **Scope and Objectives**: This section clarifies the overarching goals of the deliverable, detailing its boundaries and intended outcomes. It establishes how these objectives align with the broader project framework and the specific targets defined for WP4.

3. **Methodology**: This section explains the foundational principles and techniques employed in defining the system architecture and evaluating trustworthiness across WP4 tasks. It covers the processes, assessments, and validation methods that support the architecture's development and integration with the wider project activities.

4. **Federated Learning**: This section presents the MANOLO federated learning framework developed in Task 4.1, explaining its role in training and optimising AI models collaboratively across cloud-edge nodes while preserving data locality. It outlines the framework's design objectives—minimising communication and energy costs, coping with heterogeneous clients and non-IID data—and details its sub-modules, requirements, enabling technologies, evaluation metrics, and development timeline.

5. **Cloud-edge Resource and Infrastructure Mapping:** This section presents the progress of the infrastructure mapping component developed in Task 4.2, which abstracts and categorizes computational resources and AI assets across the distributed heterogeneous cloud-edge system infrastructure, taking into consideration different aspects of resource capabilities (processing, storage, energy, capacity, etc), networking topology and associated AI functions and assets.

6. **Trustworthy-driven Policy Validation:** This section explains the progress of the Policy Validation component developed in Task 4.3, which is responsible for ensuring that AI/ML functions and models conform to the expectations set by predefined and configurable criteria/policies (trustworthiness, performance, availability, efficiency, etc.) which will be defined and supplied by the Cloud-edge-far-edge Continuum operating system using MANOLO.

7. **Trustworthy & Infrastructure Oriented Model and Learning Allocation:** This section details the progress of the Allocation Manager component developed in Task 4.4, which aims to design and develop optimised mechanisms for allocating different AI/ML models and functions (training/inference) across the entire edge-cloud continuum.

8. **Use Case Alignment:** This section shows how each component can be used to achieve the WP4 goals in every use case, taking into consideration each use case specification.

9. **Conclusions and Next Steps:** This section summarises the M18 achievements of the WP4 and defines the roadmap to the final release.

2. Scope and Objectives

2.1 Technical Objectives

The main objectives of WP4 can be summarized as follows:

- **Objective 1**: Design and develop the main components in this module
- **Objective 2**: Identify specific needs of the proposed hyper distributed training model
- **Objective 3**: Define abstraction parameters to characterise resources
- **Objective 4**: Identify compliance policies for validation
- **Objective 5**: Develop novel resources allocation mechanism suiting specific AI/ML models training needs

These WP4 general objectives are related to the different tasks in the WP. That is, Objective 1 refers to the development and integration of all the components in WP4. Objective 2 is related to the Federated Learning (T4.1), Objective 3 is related to the mapping (T4.2), Objective 4 to the policies (T4.3), and finally Objective 5 to the optimization of resource allocation mechanisms (T4.4).

2.2 Alignment with Project-Wide Objectives

MANOLO Objective 4: *Optimise and automate the allocation of efficient AI models, functions (training and inference) and data in the Cloud-Edge continuum according to requirements and constraints of resources and infrastructures*

The whole set of processes linked to deploying AI-based strategies is highly consuming in terms of data processing, which indeed has severe implications on resource utilisation and energy consumption. This objective aims at minimising this issue by developing AI data processing and AI models training strategies leveraging the benefits brought by considering the whole edge-cloud computing continuum. Indeed, by considering such novel computing paradigm, the load required to manage and process may be smartly distributed throughout the entire computing stack, under a completely adaptive framework, considering: i) distinct AI paradigms, so that different approaches may be considered (central, distributed, hybrid, dynamic) according to the specific needs on each individual scenario, and; ii) the dynamic nature of data, in terms for example of availability, freshness, volume, and specific constraints such as, for example privacy. Consequently, the data to be forwarded to the cloud and to the edge is substantially reduced (i.e., network load reduction and lower data storage and processing needs). However, the adoption of such a highly distributed paradigm, imposes a set of research challenges and thus objectives that must be achieved, including:

- (RO4.1) creation of distributed learning strategies accommodating the specific contextual characteristics of the entire Edge-cloud layout.
- (RO4.2) a distributed data process supporting the different approaches to be considered to manage data as well as the produced models (processing, storage, preservation policies).
- (RO4.3) optimal policy-aware resources allocation in the Edge-cloud stack to properly accommodate the distributed, dynamic and hybrid end-to-end training and inference functionalities envisioned in MANOLO toward an efficient and trustworthy AI processing.

Alignment of WP4 Objectives with Project-Wide Objective 4: WP4 Objective 1, Design and develop the core components; delivers orchestrator, abstraction layer and policy validator needed for distributed learning

(RO4.1), data management workflows (RO4.2) and policy-aware placement (RO4.3). WP4 Objective 3, Define resource-abstraction parameters; Provides a machine-readable descriptor set that the scheduler uses for constraint-driven placement (RO4.3) while enabling context-aware learning topologies (RO4.1). WP4 Objective 4, Identify compliance policies for validation; embeds GDPR/AI-Act rules into data handling (RO4.2) and becomes a mandatory input to the policy-aware resource allocator (RO4.3), ensuring trustworthy execution. WP4 Objective 5, Develop novel resource-allocation mechanism; implements the optimal, policy-aware scheduling required by RO4.3 and reduces data movement / energy use, reinforcing efficient distributed learning in RO4.1. In short, every WP4 objective provides concrete methods, components or knowledge that directly realise or enable the three research objectives (RO4.1-RO4.3), thereby fulfilling Project Objective 4's ambition of automated, efficient and trustworthy AI deployment across the Cloud-Edge continuum.

MANOLO Objective 5: *Ensure AI trustworthiness and legal compliance development and operation by-design*

Following an ethically sound human-centric approach to AI and adopting the most recent approaches on bridging the gap between trustworthy AI development, as well as technical and methodological practices, MANOLO will adopt a co-design process to ensure trustworthy AI, i.e., the Z-Inspection® approach 4 (CC BY-NC-SA licence). The Z- Inspection® process is a formalised and principled approach for evaluating an AI system in design, under deployment, or already in-use, aimed at ensuring that the final system iteration is both trustworthy and trusted. It aligns with broader trends towards the design and assurance of trustworthy AI systems. This approach will allow for the thorough evaluation and validation of our AI-based framework, while also ensuring that all developed components abide by the EU Guidelines for Trustworthy AI 5, through development of socio-technical scenarios that will highlight any ethical issues and tensions, monitored by clear quantitative metrics. In addition, the MANOLO framework will be designed taking into relevant legal aspects (AI, Data and Data Governance Acts, GDPR, etc), delivering a framework that fits current required specifications. At the same time, the MANOLO's Ethics Advisory Board (EAB), will ensure that all ethical and legal aspects are carefully considered throughout MANOLO, and especially when focusing on the project implementation and use cases' deployment. Final results will be evaluated and validated through a key experts' workshop, to ensure alignment with the European Approach to AI. A handbook will also be created to support the development as well as future exploitation.

Alignment of WP4 Objectives with Project-Wide Objective 5:

WP4-O1: Design and develop core components ➜ Builds the WP4 modules that allows embedding Z-Inspection® findings and EU Trustworthy-AI guidelines directly in the runtime stack to assure trustworthy AI allocation.

WP4-O2: Identify needs of the hyper-distributed training model ➜ Captures ethical, privacy and legal constraints up-front so they become hard requirements during co-design and later Z-Inspection® assessments.

WP4-O3: Define resource-abstraction parameters ➜ Adds security, data-sovereignty and provenance tags to every resource descriptor, enabling traceable, policy-aware deployments across the continuum.

WP4-O4: Identify compliance policies for validation ➜ Translates GDPR, AI-Act and sectoral rules into machine-readable policies enforced at design-time and run-time, directly addressing trustworthiness and legal conformity.

WP4-O5: Develop novel resource-allocation mechanism ➜ Implements resource allocation methods that obeys the above policies, guarantees lawful data/model placement, and produces auditable logs.

2.3 Expected Outcomes and Indicators

According to Objective 4 in the previous section, the main goal is to reduce the resource and energy demands of AI by using edge-cloud computing to distribute processing tasks. This approach adapts to different AI models and data types, reducing network load. Key challenges include creating distributed learning strategies, managing data and models, and optimizing resource allocation across the system. In alignment with this Objective 4 in the DoA, MANOLO has set some indicators as shown in Table 1. The target values are the ones proposed at the end of the project and the progress is the current status of these indicators.

As shown, there is progress in practically all the indicators, except in Indicator 2. In addition, most indicators show progress, especially related to Federated Learning. The main reason is that development and research of Federated Learning could be carried out independently, as it is a special case of resource allocation. With MANOLO's FL setup, we have shrunk the model, let each device train a bit longer before syncing, and talked only to the clients that add real value. Thanks to those changes, the demo is exceeding its targets: compute-side energy is down by 63.2 % (goal > 25 %), network traffic is down by 90.7 % (goal 25%), and training completes 69.2% faster (goal > 40%). In addition, four new resource-allocation methods have been delivered, three of which were built just for FL.

Progress on new resource allocation algorithms for both inference and training now requires further integration of work developed in tasks T4.2 (mapping), T4.3 (policies), and T4.4 (resource allocation). MANOLO, currently, has a basic resource allocation mechanism that involves these tasks, which essentially validates the node where the experiment will be conducted, provided by the AI Workload Orchestrator.

In the second part of the project, as will be described in the plans for tasks T4.2, T4.3, and T4.4, new optimization mechanisms for resource allocation will be investigated using AI. In this case, different nodes will be suggested to the AI Workload Orchestrator for conducting the experiment, as well as new policy reinforcement actions.

Table 2. Objective 4 Indicators

#	Indicator	Target Value	Progress
1	Overall energy reduction in computing processing in cloud-edge settings:	>25%	63.2% (FL)
2	Combined energy reduction in model training plus re-training activities:	x>40%	Not yet considered
3	Network traffic reduction	25%	90.7 % (FL)
4	Novel algorithms for resources allocation in the cloud-edge suiting specific data scenarios	>3	3 (FL)+ 1 (T4.2+T4.4)
5	Training time reduction	>40%	69.2% (FL)

Apart from these more technical indicators, there are other indicators related to the previously described MANOLO Objective 5. This objective argues that MANOLO will use the Z-Inspection® method to ensure ethically sound, trustworthy AI, in line with EU regulations; also, that an Ethics Advisory Board will oversee

the process, and final results will be validated by experts. Finally, a handbook will also be created to support future development and use. The indicators of this MANOLO Objective 5 are actually indicators of MANOLO's entire framework, and then different WPs will contribute to them. These indicators will be assessed at the end of the project. At this stage of the project two workshops have been organized, one about the Z-inspection tool, and another on the Neuromorphic chip with more than 50 participants.

3. Methodology

3.1 General Architecture

The *"Trustworthy & Infrastructure Oriented Model and Learning Allocation"* (also called the Cloud-Edge-Continuum (CEC) model/resource allocation module) represented by WP4 in MANOLO is part of the functionalities provided by the MANOLO Library described in deliverable D1.2 "MANOLO Architecture, and Benchmarking framework v.1" (ref). Apart from the main functionalities implemented in the core module, the MANOLO Library also provides some extra functionalities, such as this CEC model/resource allocation (Other modules in Figure 1). In addition to having the functionalities in the library, these can also be accessed separately by developers to be integrated into their AI pipelines if there is a need.

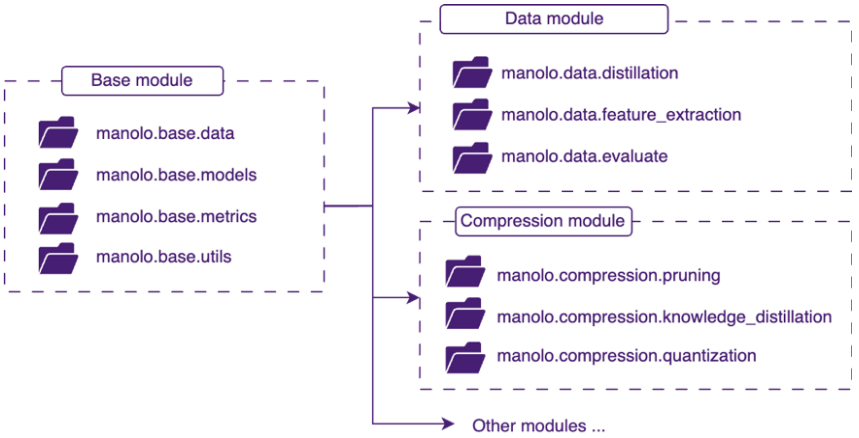
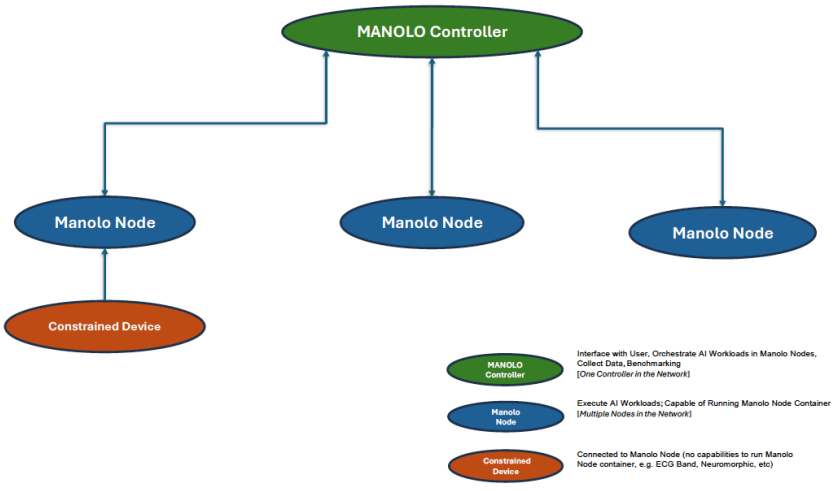


Figure 1. MANOLO Library

The *"Trustworthy & Infrastructure Oriented Model and Learning Allocation"* is part of the MANOLO controller, one per cluster, described in deliverable D1.3 "MANOLO Architecture, and Benchmarking framework v.2", see Figure 2, where it is called Infrastructure & Policies.



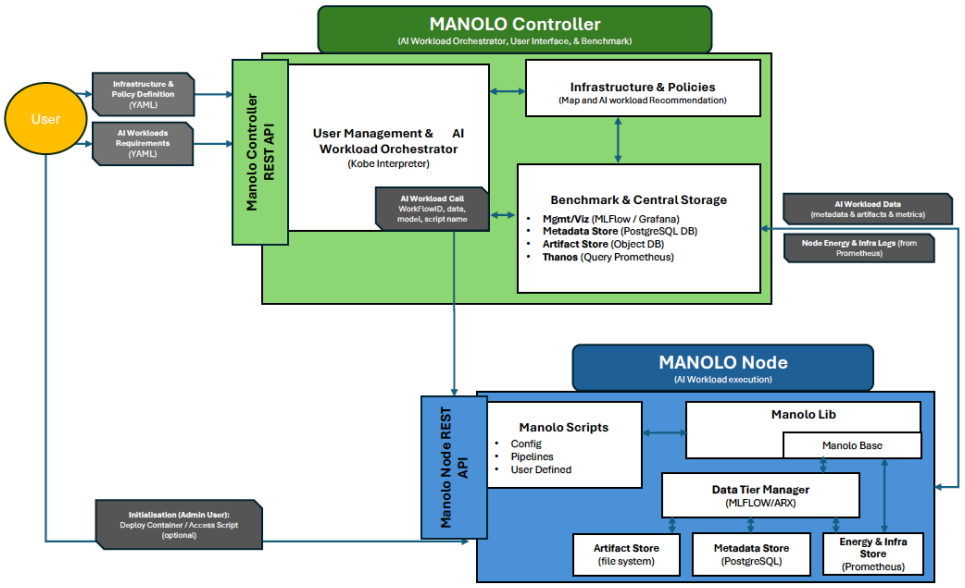


Figure 2. MANOLO network and system architecture

This functionality manages the selection of resources for the allocation of the different AI/ML models (AI Workloads) for both phases training and inference, spanning across the entire edge-cloud continuum, by applying optimised mechanisms to select the most suitable resources according to i) user requirements, ii) model requirements, iii) the landscape (mapping of the available resources), iv) the optional Federated Learning, v) and the performance from previous executions/experiments provided by the Benchmarking and the Monitoring modules.

The *"Trustworthy & Infrastructure Oriented Model and Learning Allocation"* also includes the MANOLO's Policy Compliance Manager for the edge-to-cloud continuum, which ensures that AI/ML models conform to the expectations set by predefined and configurable policies (trustworthiness, performance, availability, efficiency, etc.) at runtime. The final output to the user is in the form of an alert as well as a suggestion of enforcement action. The MANOLO Developers also can create, update or delete these policies to be applied.

The general architecture of WP4 is shown in Figure 3.

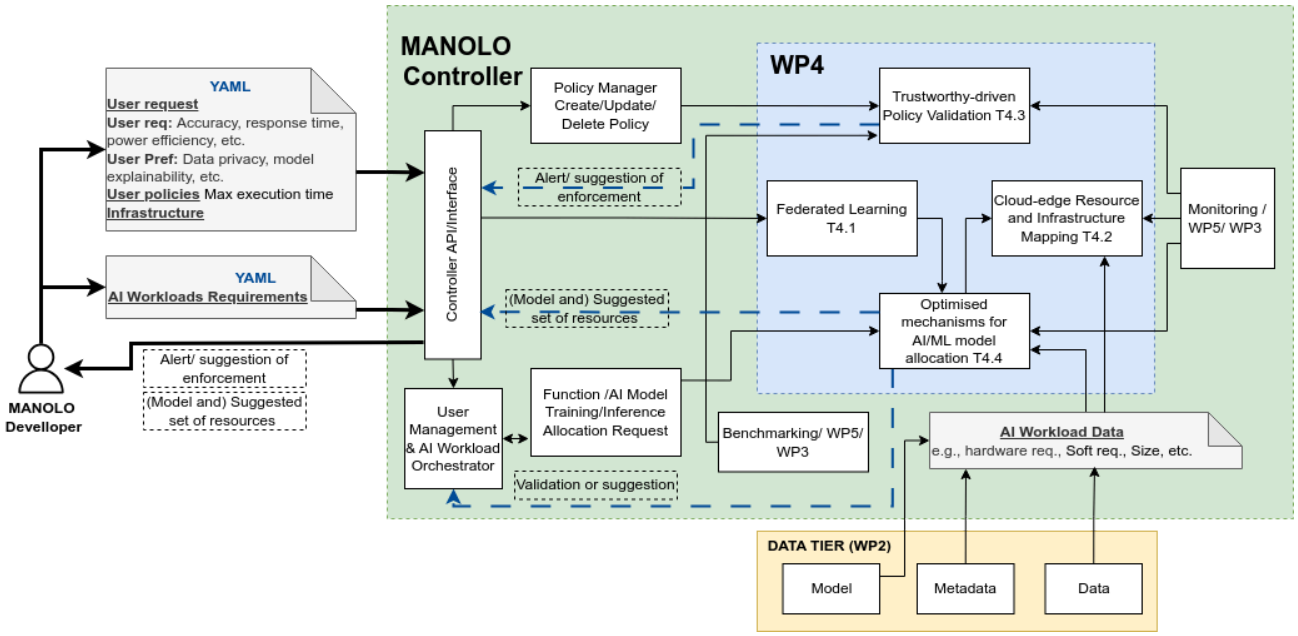


Figure 3. WP4 architecture

3.2 Implementation and Validation Steps

This section provides a practical procedure to use the results of WP4 through a simple "Hello World" example. The goal is to demonstrate how the different services can be integrated into a typical development workflow, highlighting key implementation steps such as setup, API interaction, and basic validation.

By following this example, MANOLO Developers can quickly understand how to define and register an experiment, computational resources and actions, interact with the API endpoints, and verify that the system behaves as expected. The example serves as a foundation for more complex use cases and is intended to validate the core functionality of the WP4 microservice architecture in a controlled, reproducible manner.

To support this, we created two different simple validation workflows. The first workflow is relying on the requirement and infrastructure input from the MANOLO Developer; the second workflow is using the monitoring tool to get the information about the requirements for functions/actions and the available topology from the system itself.

Hello World Flow 1:

1. The infrastructure and policy information are updated by the MANOLO developer, by sending that information to the MANOLO API.
2. The MANOLO API forwards that information to the Allocation Manager (T4.4.).
3. The Allocation Manager (T4.4.) forwards that information to the Landscape Manager Allocation Manager (T4.2.).
4. The MANOLO Developer specifies an experiment in a YAML file, the AI workload requirements, that is sent to the Controller API.
5. The AI workload requirement YAML file gets forwarded from the Controller API to the User Management & AI Workload Orchestrator.

6. The User Management & AI Workload Orchestrator needs to validate the possibility of deployment of this request. For that a validation request is sent to the Allocation Manager (T4.4.).
7. The Allocation Manager (T4.4.) sends a validation request to the Landscape Manager (T4.2.).
8. The Landscape Manager (T4.2.) checks against its database of function requirements and topology information, if it is possible to deploy the experiment and confirms that with the Allocation Manager (T4.4.).
9. The Allocation Manager (T4.4.) gets periodically updated by the FL manager (T4.1.) with the newest information about federated learning.
10. By aggregating all this information, the Allocation Manager (T4.4.) can make a sophisticated decision about the deployment and can confirm this information back to the AI Workload Orchestrator.
11. The AI Workload Orchestrator deploys the experiment on the requested nodes through the MANOLO Node API.
12. The MANOLO Developer gets information from the Policy Manager (T4.3.) about the set policy and if these policies are violated, which is forwarded to the MANOLO developer.

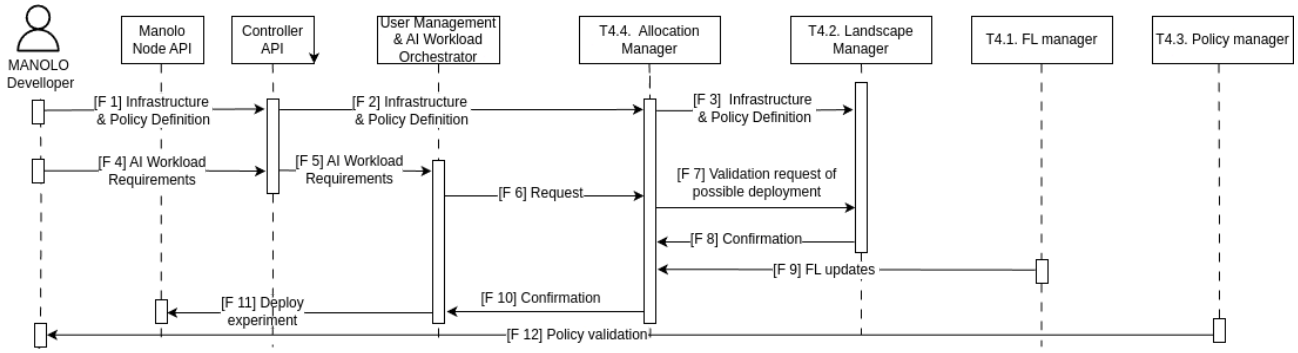


Figure 4. Hello world flow 1: requirement / infra input from MANOLO Developer

Hello World Flow 2:

1. Infrastructure updates sent from the Monitoring Tool (WP3/WP5) to the Landscape Manager (T4.2.) to update the topology that is tracked automatically.
2. AI Workload updates are sent from the Requirement Tool (WP2) to the Landscape Manager (T4.2.) to update the requirements for the specific AI Workloads automatically
3. The user specifies an AI Workload Requirement in a YAML file that is sent to the Controller API.
4. The AI Workload Requirement file gets forwarded from the Controller API to the User Management & AI Workload Orchestrator.
5. User Management & AI Workload Orchestrator needs to validate the possibility of deployment of this request. For that a validation request is sent to the Allocation Manager (T4.4.).
6. The Allocation Manager (T4.4.) sends a validation request to the Landscape Manager (T4.2.).
7. The Landscape Manager (T4.2.) checks against its database of function requirements and topology information, if it is possible to deploy the experiment and confirms that with the Allocation Manager (T4.4.).
8. The Allocation Manager (T4.4.) gets periodically updated by the FL manager (T4.1.) with the newest information about federated learning.

9. By aggregating all this information the Allocation Manager (T4.4.) can make a sophisticated decision about the deployment and can confirm this information back to User Management & AI Workload Orchestrator.
10. User Management & AI Workload Orchestrator deploys the experiment on the requested nodes through the MANOLO Node API.
11. The MANOLO Developer gets information from the Policy Manager (T4.3.) about the set policy and if these policies are violated, which is forwarded to the MANOLO developer.

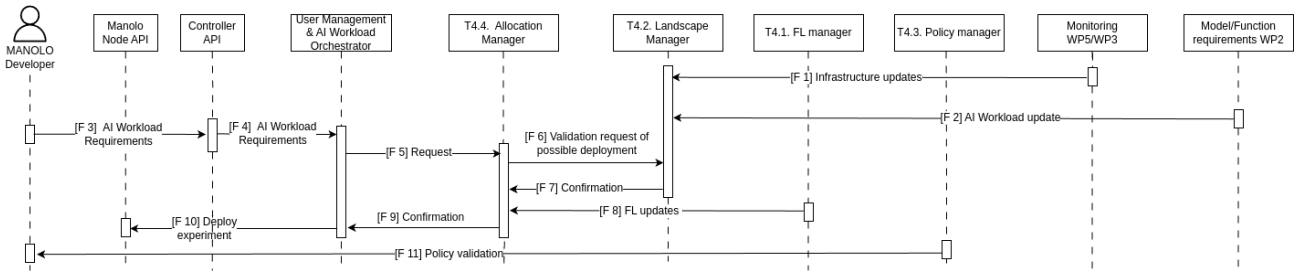


Figure 5. Hello world flow 2: requirement / infra input from monitoring

4. Federated Learning

4.1. Introduction and Contextual Framework

The MANOLO project aims to build better algorithms and tools for AI systems that work across cloud and edge environments. Task 4.1 specifically tackles Federated Learning (FL) to solve computational challenges in distributed systems. We've created an advanced federated learning framework under Objective 4, which focuses on optimizing AI models and functions in the Cloud-Edge continuum based on resource constraints.

Our project proposal outlined the need to develop components that map requirements for distributed training models while cutting down on data communication and energy use between nodes. We've focused on handling different types of clients, optimizing models for edge computing, measuring energy consumption, and dealing with non-IID data - all crucial for real-world federated learning deployments.

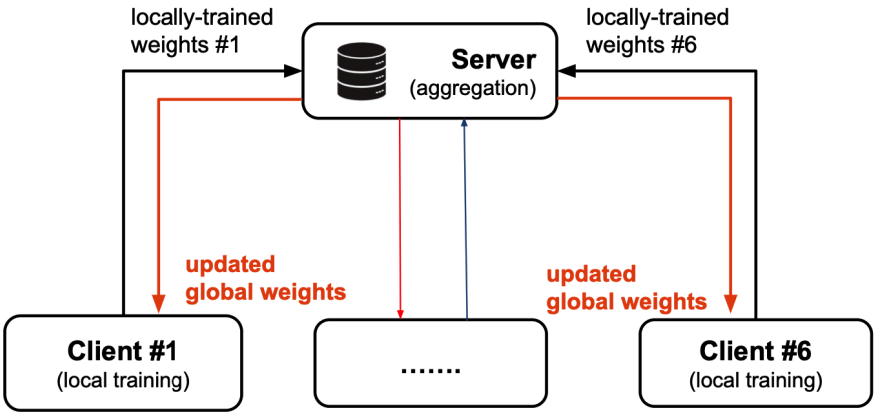


Figure 6. FL architecture

4.1.1 Bitbrain Use Case: Dataset and Implementation Context

We worked with Bitbrain's scenario, whose EEG headband technology provides our data. Their dataset includes EEG recordings from 127 subjects, with each subject's data covering a full night's sleep - making it ideal for sleep classification research. For our specific implementation, we're using data from six subjects, though the full dataset is much larger. Our federated learning system is designed for Bitbrain's "Scenario 2 - MANOLO suite" deployment, where computation happens on tablet devices rather than on the headbands themselves.

The setup works as follows: the EEG headband collects brain signals and sends them via Bluetooth to a tablet, which does the actual computing. These tablets have more processing power than the headbands but still face resource limitations compared to cloud servers. That's why we're using an efficient model adapted from Task 5.3's explainability work. Our implementation follows this process:

1. Load a pre-trained model onto the tablet.
2. The headband collects and transmits EEG data to the tablet via Bluetooth.
3. After the sleep session, we process and anonymize the data.

4. Then we use Federated Learning to improve models across multiple patient devices while keeping data private.

This setup represents exactly the kind of cloud-edge balance MANOLO tries to solve - keeping sensitive patient data private (by keeping it local), running efficiently on limited-resource tablets, and improving models through FL across different patients' data. Using tablets with Bluetooth connections gives us a practical middle ground that protects privacy while enabling advanced model training without needing continuous cloud connections during data collection.

4.2. Methodological Framework

4.2.1 Federated Learning Architecture

Our system uses a server-client setup with the Flower framework, designed specifically for EEG sleep stage classification - an area with heavy computational needs and privacy concerns. It's a perfect example of where federated learning makes sense for healthcare.

As shown in Figure 6, a central server coordinates the process using advanced parameter aggregation, with six separate clients representing individual data sources. The server gets locally-trained weights from each client, smartly combines these parameters, and sends updated global weights back to everyone. This approach preserves privacy while enabling collaborative improvement.

4.2.2 Advanced Non-IID Data Handling

One of the biggest challenges in FL is when clients have non-identical data distributions. Looking at our dataset, we found major differences across the six subjects, as Figure 7 shows.

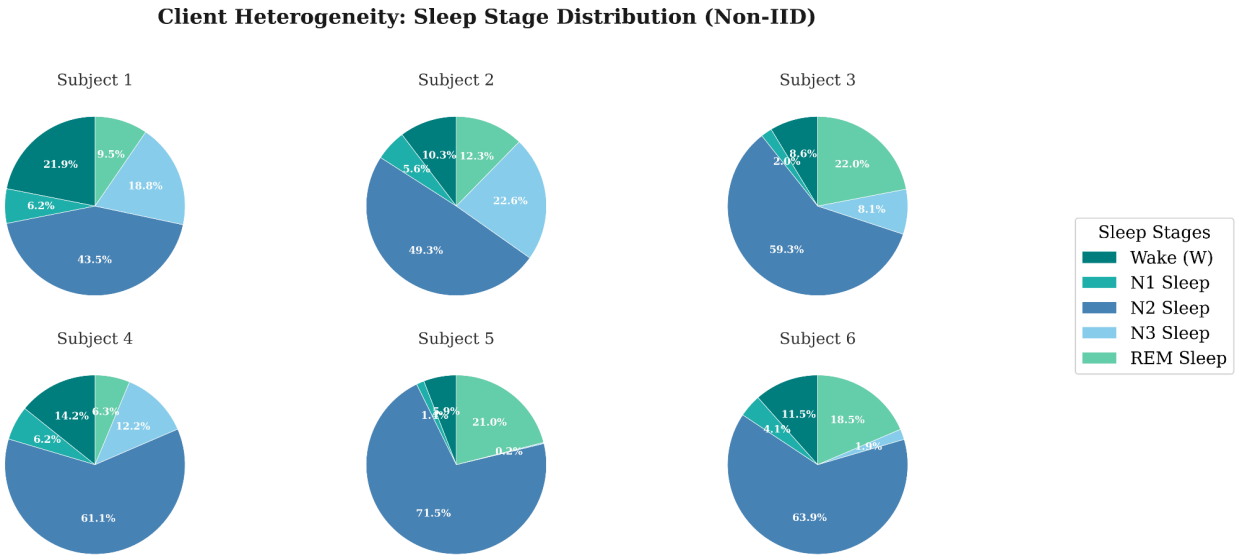


Figure 7. Visualization

The visualization in Figure 7 quantitatively demonstrates the profound heterogeneity in sleep stage distribution across the six participants. Most notably, Subjects 4, 5, and 6 exhibit marked predominance of N2 Sleep (>60% of samples), while Subjects 1, 2, and 3 manifest considerably more balanced distributions. This heterogeneity constitutes a paradigmatic exemplification of non-IID data in real-world clinical contexts, precisely the challenge our FL implementation must overcome.

Our approach addresses this distributional heterogeneity through several sophisticated mechanisms:

1. **Class-Aware Parameter Aggregation**: We implemented a novel aggregation strategy that dynamically weights client contributions based on class representation quality and performance metrics. This strategy applies differentiated weighting to model parameters during aggregation, prioritizing clients with superior performance on specific classes, particularly those underrepresented globally.
2. **Adaptive Focal Loss Function**: To address extreme class imbalance, we developed an AdaptiveFocalLoss mechanism that dynamically adjusts alpha values based on evolving class distributions. This approach provides significantly enhanced performance in environments with highly skewed class distributions.
3. **Target-based Resampling**: Each client implements advanced rebalancing for local training using median class size to determine target sample distributions per class.

4.3. Implementation Details

4.3.1 Scaling to Six Heterogeneous Clients

Our system successfully scales to six federated clients, each with its own EEG sleep data and different distributions. Table 3 shows how we configured these different devices, with resources carefully set to match real-world edge computing limitations.

Table 3. FL configurations

Component	Device	CPU Requests	CPU Limits	Memory Requests	Memory Limits
fl-server	Server	2 CPU	4 CPU	6 Gi	8 Gi
fl-client-1	Raspberry Pi	1 CPU	2 CPU	3 Gi	6 Gi
fl-client-2	Raspberry Pi	1 CPU	2 CPU	3 Gi	6 Gi
fl-client-3	Raspberry Pi	1 CPU	2 CPU	3 Gi	6 Gi
fl-client-4	Jetson	0.75 CPU	2 CPU	3 Gi	4 Gi
fl-client-5	Jetson	0.75 CPU	2 CPU	3 Gi	4 Gi
fl-client-6	Jetson	0.75 CPU	2 CPU	3 Gi	4 Gi

We're using three Raspberry Pi devices (clients 1-3) and three Jetson devices (clients 4-6), each with different CPU and memory allocations. This mix matches real-world edge environments where devices vary in power. The server gets more resources (2-4 CPUs, 6-8 Gi memory) to handle the parameter aggregation and global updates.

4.3.2 Heterogeneous Device Integration

Our implementation works with various device capabilities by:

1. **Resource-Aware Client Setup**: Clients automatically adjust batch sizes and memory use based on what they have available. As you can see, Jetson devices (clients 4-6) run with less CPU (0.75 CPU requests) than Raspberry Pis (1 CPU request), so we adjust for that.

2. **Flexible Model Design**: The model architecture adapts to available resources by using:

    - SeparableConv1D layers instead of standard convolutions
    - Fewer filters in deep layers
    - Squeeze-and-excitation attention blocks that work well despite having fewer parameters
    - Smaller dense layers

3. **Deployment Optimizations**: We use Lite conversion with post-training quantization, which shrinks model size by over 67%, making it possible to run on various edge devices with limited resources.

4.3.3 Energy Consumption Evaluation

We've thoroughly analyzed energy consumption, addressing a key performance target from the MANOLO project proposal. Our system tracks:

1. **Component-Level Energy**: Detailed measurement of energy use across CPU, memory, and other components.
2. **Communication Energy Costs**: Measuring energy used for transmitting model parameters.
3. **Comparative Benchmarking**: Comparing centralized versus federated approaches, showing major efficiency improvements as illustrated in Figure 5.

4.3.4 Hyperparameter Tuning

Our implementation includes sophisticated tuning mechanisms:

1. **Parameter Adaptation**: Dynamically adjusting learning rates based on how clients are performing.
2. **Proximal Term Regularization**: Using FedProx extensions with adjustable proximal terms to reduce client drift.
3. **Class Weight Optimization**: Automatically calibrating class weights as distribution patterns change.

4.4. Experimental Results

4.4.1 Non-IID Handling Performance

The effectiveness of our non-IID handling mechanisms is demonstrated in Figure 8, which illustrates the impact of specialized non-IID handling on classification accuracy.

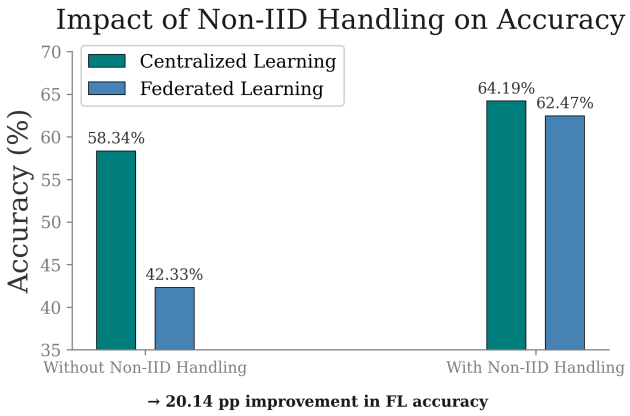


Figure 8. Impact of Non-IID Handling on Accuracy

As shown in Figure 8, without special handling, federated learning performs much worse than centralized approaches (42.33% vs. 58.34%). But with our non-IID handling techniques, federated learning nearly matches centralized approaches (62.47% vs. 64.19%) - that's a substantial 20.14 percentage point improvement!

This improvement highlights how effective our class-aware aggregation and adaptive focal loss are at solving the statistical heterogeneity problem.

4.4.2 Training Efficiency

Figure 9 shows the impressive training speed improvements from our federated approach.

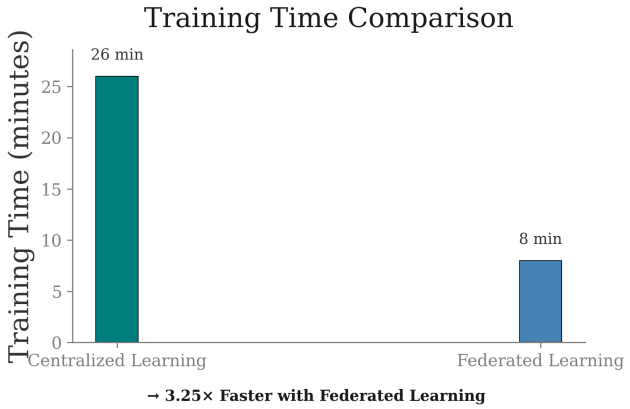


Figure 9. Training time results

As Figure 9 demonstrates, our distributed training is 3.25× faster than equivalent centralized approaches (8 minutes vs. 26 minutes). This speed boost comes from:

1. **Parallel Processing**: Computing simultaneously across multiple clients
2. **Smaller Per-Client Workloads**: Each client handles less data, so they converge faster
3. **Optimized Communication**: Minimizing parameter exchange overhead

The speed improvement is particularly impressive given the varied computing power of our devices, as detailed in Table 3.

4.4.3 Energy Consumption Reduction

Figure 10 presents a comparison of energy consumption between centralized and Federated Learning paradigms.

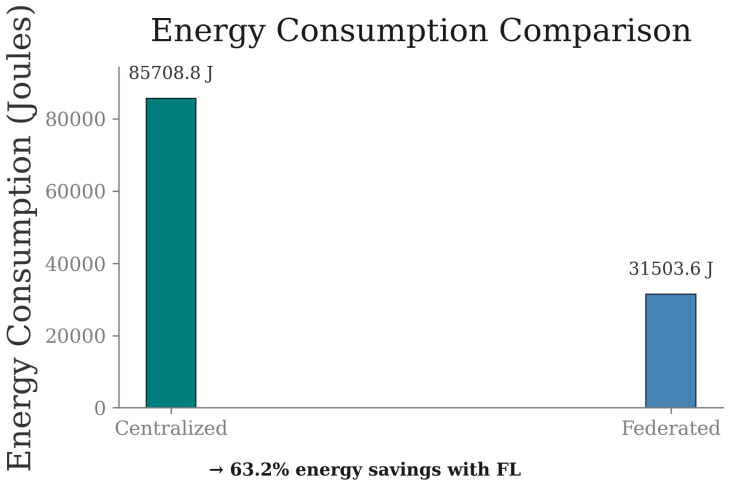


Figure 10. Energy consumption

It shows our federated learning implementation achieves remarkable energy savings - 63.2% less energy than centralized approaches (31,503.6 J vs. 85,708.8 J). This reduction far exceeds the 25% target set in the MANOLO project proposal.

These energy savings are especially significant because client devices have much less computing power than centralized servers. The Raspberry Pi and Jetson devices (shown in the Table) typically use less peak power but achieve much better energy efficiency through distributed computing and local processing optimization.

4.5. Alignment with Project Objectives

Our implementation exceeds the project's key performance targets:

1. **Energy Reduction**: Achieved 63.2% energy savings in computing (target was >25%)
2. **Training Speed**: Achieved 69.2% reduction (3.25× faster) in training time (target was >40%)
3. **Edge Model Optimization**: Cut model parameters from ~200,000 to ~14,500 (93% reduction) while maintaining similar accuracy

4. **Network Traffic**: Implemented efficient parameter exchange that significantly reduces communication overhead

These results show that our implementation not only meets but goes well beyond the targets set in the MANOLO project proposal.

4.6. Summary & Next Steps

The Task 4.1 implementation represents a significant advancement in distributed AI optimization for cloud-edge environments. We've successfully tackled the major challenges of federated learning - non-IID data handling, energy efficiency, and computational optimization for different edge devices.

Our experimental results show exceptional performance across multiple dimensions, exceeding project targets for energy reduction and training efficiency while maintaining good classification accuracy. Successfully scaling to six heterogeneous clients with varying computational capabilities (Raspberry Pi and Jetson devices) further proves our implementation works in realistic deployment scenarios.

Our next steps are to investigate progressive transfer learning for new clients, aiming to achieve faster convergence, which translates to reduced training time and energy consumption. Additionally, we plan to scale the system to support over 10 concurrent clients to better reflect a realistic deployment scenario.

4.7. References

1. McMahan, B., et al. (2017). Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics (pp. 1273-1282).
2. Li, T., et al. (2020). Federated optimization in heterogeneous networks. Proceedings of Machine Learning and Systems, 2, 429-450.
3. Wang, H., et al. (2021). Federated learning with matched averaging. In International Conference on Learning Representations.
4. Kairouz, P., et al. (2021). Advances and open problems in federated learning. Foundations and Trends in Machine Learning, 14(1-2), 1-210.
5. Xia, W., et al. (2020). Mobile edge cloud-based industrial internet of things: improving edge intelligence with hierarchical SDN controllers. IEEE Vehicular Technology Magazine, 15(1), 36-45.

5. Cloud-edge Resource and Infrastructure Mapping

5.1 Technical Requirements

This component has following technical requirements:

Table 4. Technical requirements for T4.2

Programming Language	• Python Version: 3.9 or newer.
Required Libraries	• Fast API • Uvicorn • Motor • Python-multipath
Required Technologies	• MongoDB docker image (first Iteration) • Thanos integration (second Iteration)
Input Requirements	• Preferred node to compute on • Function that should be executed
Output	• Confirmation that the action/function can be executed on the specified node (first Iteration) • List of nodes that are able to execute the action/function (second Iteration)
Dependencies for Landscape Manager	• Monitoring (WP3/WP5) must provide topology information to keep the Landscape Manager up-to-date for the available resources • Function requirements (WP2) must provide function information to keep the Landscape Manager up-to-date for the available functions • For testing purposes, the Landscape Manager can be manually updated on newly available nodes and functions • T4.4 must call the Landscape Manager in order to get the validation if an experiment is able to be deployed
Containerization (Optional for Future Integration)	• The component can be containerized using Docker for consistent deployment and scalable integration with other components. • It can be exposed and integrated with other components either via API calls or as a Dockerized service.

5.2 Component Description

The Landscape Manager is a central architectural component responsible for maintaining a comprehensive and up-to-date representation of the system's infrastructure, benchmarks and AI-related functional requirements. This component operates as a repository and coordination hub, aggregating data originating from distinct subsystems within the architecture. Specifically, information about the topology is supplied by the WP3/WP5 Monitoring component and information about the actions/functions is supplied by WP2.

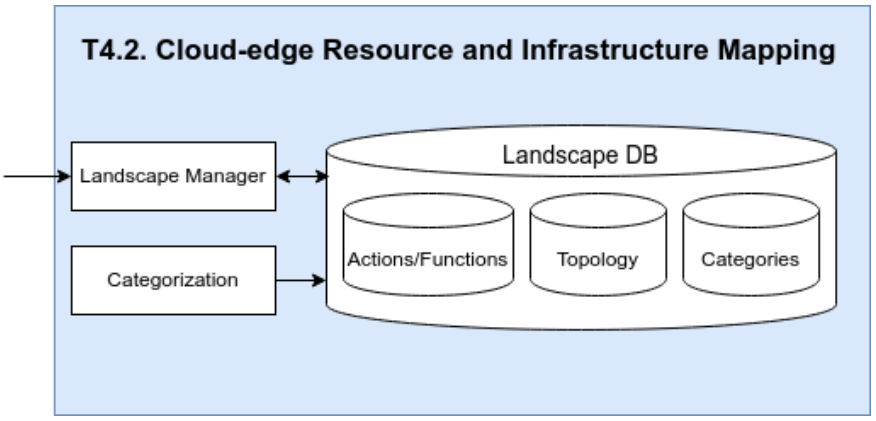


Figure 11. T4.2 architecture

The Landscape Manager is the front door of the cloud-edge resource and infrastructure mapping component. It is an API programmed in Python3 using the FastAPI library. All communication to the underlying sub-components is through this API. Incoming information about the topology and about the actions/functions is passed to the Landscape DB and stored in a respective data storage, while this new information is triggering the Categorization sub-component.

First Iteration:

- Landscape DB: Independent MongoDB deployment that keeps track of all information
    - Will be updated through the API

Second Iteration:

- Thanos: Integrated access to Thanos that will monitor the entire system and will function as the central DB for the entire Manolo-Project

All received data is persistently stored within the Landscape DB, which serves as an integrated knowledge base. The information about the actions/functions is stored separately from the topology information. Whenever there is new information the Categorization tool will be triggered. The Categorization tool will do an initial, static mapping, which actions/functions can be executed on which nodes. This information will be stored inside the CategoriesDB inside the database.

When the Landscape Manager is now called by T4.4. to give back a list of nodes which are able to run the action/function at question, it will be just a single call to the CategoriesDB to reply with the mapping.

5.3 Workflow

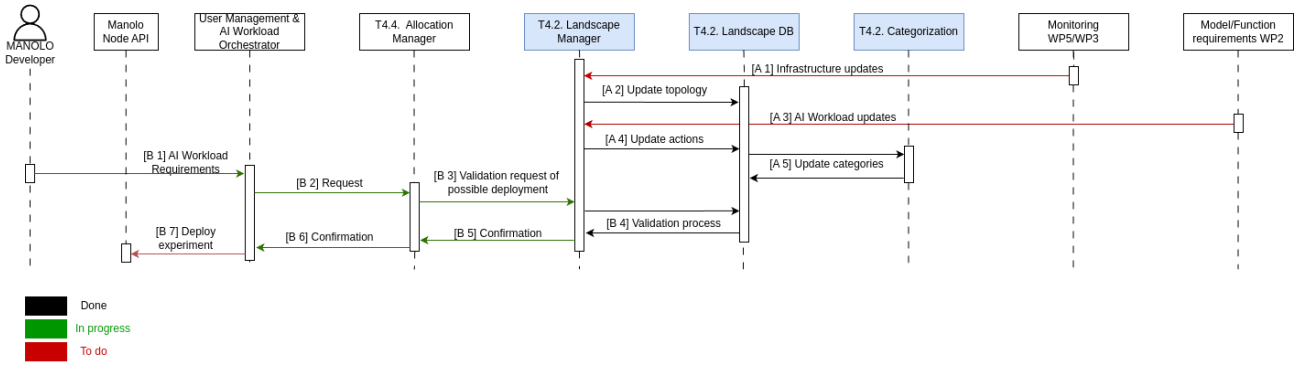


Figure 12. T4.2 workflow

In Figure 12 the entire Workflow of the cloud-edge resource and infrastructure mapping component is shown, together with the current state of integration. The internal communication of our tool in T4.2. is already fully integrated and done, while the communication with T4.4. is about to be finalized as well. T4.4. is working on establishing the connection to the AI Workload Orchestrator and over this to the MANOLO Developer.

Workflow A:

1. The monitoring tool of WP5/WP3 is calling an API endpoint of the Landscape Manager to update the Landscape DB, more precisely the topology information that is stored there.
2. Through the API of the Landscape Manager the Landscape DB is updated.
3. The model/function requirements from WP2 get updated and send that information to an API endpoint of the Landscape Manager.
4. The information will be stored in the Landscape DB, inside the actions/function store.
5. The Categorization component is triggered whenever a new update to the topology or to the action function store is occurring.
    a. The order of the events [A.1], [A.2] or [A.3], [A.4] doesn't matter
    b. The available nodes will be compared to the resource constraints and will result in a mapping of possible nodes for deployment to actions/functions that can be deployed.
    c. This information is stored in the Landscape DB, inside the categorization component.

Workflow B:

1. The MANOLO Developer initiates an experiment by sending the AI Workload Requirements to the User Management & AI Workload Orchestrator
2. The User Management & AI Workload Orchestrator will call to verify the possibility of the deployment and suggestions for deployments by calling T4.4.
3. T4.4. will make an API call to the Landscape Manager, to get information of the nodes that are able to execute the action/function in question
4. The Landscape Manager will make an internal call to the Landscape DB with the information about the function. The Landscape DB will return information about all nodes that are able to execute this action/function.
5. The Landscape Manager will return a JSON file with all nodes that are able to execute this function.

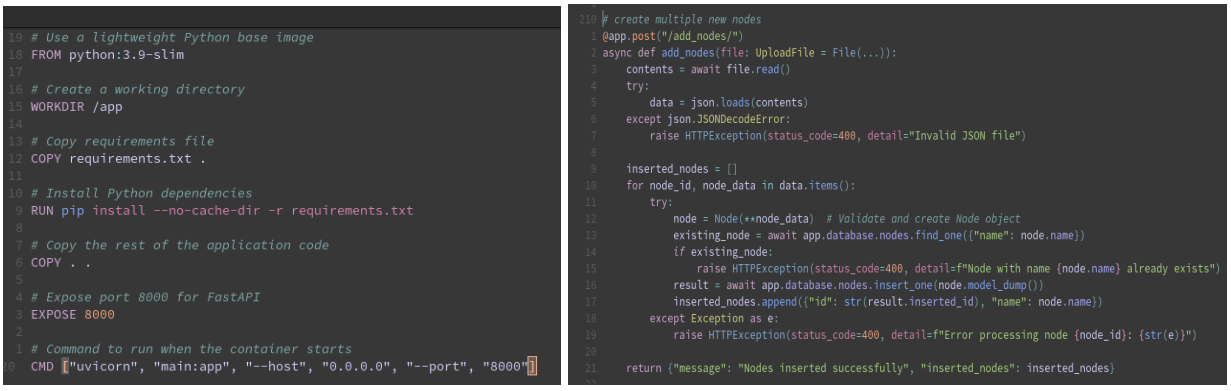
6. T4.4. will return the information about confirmation or refusal to the User Management & AI Workload Orchestrator.
7. The User Management & AI Workload Orchestrator will call the Manolo-Client API of the corresponding node to deploy the experiment.

5.4 Implementation

For the implementation of the Landscape Manager, we are using FastAPI. FastAPI is a modern, high-performance web framework for building APIs with Python, based on standard Python type hints, which enables automatic generation of OpenAPI documentation and supports asynchronous programming paradigms for scalable request handling. Its design emphasizes developer productivity and runtime efficiency by leveraging Pydantic for data validation and Starlette for asynchronous web operations. The implemented API exposes multiple endpoints that allow the insertion of single or multiple nodes into the Landscape database, as well as retrieval of stored node information, either collectively or individually, using various identifiers. Furthermore, the API supports the deletion of node entries, with options for targeted removal of specific nodes or a complete reset of the database. An equivalent set of endpoints has been developed to manage and track AI model and functional requirements, following the same interface principles as those used for infrastructure topology components.

For the Landscape DB we are using MongoDB in this first iteration and the motor Python library. MongoDB is a NoSQL, document-oriented database designed to store and manage data in a flexible, JSON-like format called BSON. Unlike traditional relational databases, MongoDB does not use tables and rows but instead organizes data into collections of documents, enabling dynamic schemas and hierarchical data structures. Motor is the official asynchronous Python driver for MongoDB, built on top of Tornado or asyncio, and designed for non-blocking database operations in modern Python applications. It enables seamless integration of MongoDB queries within asynchronous frameworks, allowing developers to perform high-throughput I/O operations without blocking the event loop. Motor closely mirrors the syntax of the synchronous PyMongo driver.

The code for the cloud-edge resource and infrastructure mapping component is available at GitLab under the open-source MIT license, see https://gitlab.com/bensalem/landscape-manager.



```
# Use a lightweight Python base image
FROM python:3.9-slim

# Create a working directory
WORKDIR /app

# Copy requirements file
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code
COPY . .

# Expose port 8000 for FastAPI
EXPOSE 8000

# Command to run when the container starts
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

```
# create multiple new nodes
@app.post("/add_nodes/")
async def add_nodes(file: UploadFile = File(...)):
    contents = await file.read()
    try:
        data = json.loads(contents)
    except json.JSONDecodeError:
        raise HTTPException(status_code=400, detail="Invalid JSON file")

    inserted_nodes = []
    for node_id, node_data in data.items():
        try:
            node = Node(**node_data)  # Validate and create Node object
            existing_node = await app.database.nodes.find_one({"name": node.name})
            if existing_node:
                raise HTTPException(status_code=400, detail=f"Node with name {node.name} already exists")
            result = await app.database.nodes.insert_one(node.model_dump())
            inserted_nodes.append({"id": str(result.inserted_id), "name": node.name})
        except Exception as e:
            raise HTTPException(status_code=400, detail=f"Error processing node {node_id}: {str(e)}")

    return {"message": "Nodes inserted successfully", "inserted_nodes": inserted_nodes}
```

Figure 13. Docker File and example of API endpoint ('add_nodes')

We use Docker to containerize the Landscape Manager, with the Dockerfile shown in Figure 13. This Dockerfile defines a minimal and portable container environment for the LandscapeManager FastAPI

microservice. It uses the lightweight `python:3.9-slim` base image to reduce image size and improve performance. The application's dependencies are installed via `requirements.txt`, and the complete source code is copied into the `/app` working directory.

The container exposes port 8000 and runs the FastAPI application using Uvicorn (`main:app`) with network access enabled via `--host 0.0.0.0`. This setup enables consistent deployment across local and cloud environments and supports integration with CI/CD pipelines or orchestration tools like Docker Compose.

To better understand the implementation of the Landscape Manager API, we present in Figure 13 an example API endpoint. The `/add_nodes/` endpoint allows bulk creation of nodes by uploading a structured JSON file. It is implemented as an asynchronous POST route and serves to efficiently register multiple computing resources in the LandscapeManager system.

The endpoint accepts a JSON file via multipart upload, parses and validates each node entry using a Pydantic model, and inserts the valid records into a MongoDB collection. Duplicate checks are performed based on the node name to prevent conflicts. If all entries are valid, a confirmation message is returned along with the database-generated IDs and names of the inserted nodes. Errors in file format or data structure are handled gracefully with clear HTTP 400 responses.

This endpoint streamlines resource registration and supports automated onboarding of multiple nodes in cloud-edge infrastructures.

The Landscape Manager FastAPI offers multiple endpoints. These endpoints facilitate CRUD operations and requirement checks to support dynamic model allocation.

General Endpoints:

- **Home:** Endpoint to verify that the Landscape Manager is up and running.
- **Documentation:** Shows all available endpoints with a description and example calls.

Node Management Endpoints:

- **Add a Single Node:** Allows the creation of a new node by submitting its specifications.
- **Bulk Add Nodes:** Enables uploading multiple nodes at once via a JSON file.
- **Retrieve All Nodes:** Fetches a list of all registered nodes.
- **Retrieve Node by ID:** Obtains details of a specific node using its unique identifier.
- **Retrieve Node by Name:** Fetches node information based on its assigned name.
- **Delete Node by ID:** Removes a node from the database using its ID.
- **Delete Node by Name:** Deletes a node identified by its name.
- **Delete All Nodes:** Clears all node entries from the system.

Function Management Endpoints:

- **Add a Single Function:** Registers a new AI function with its requirements and constraints.
- **Bulk Add Functions:** Allows the addition of multiple functions through a JSON file upload.
- **Retrieve All Functions:** Lists all available AI functions in the system.
- **Retrieve Function by ID:** Gets details of a specific function using its unique ID.
- **Retrieve Function by Name:** Fetches information about a function based on its name.

- **Delete Function by ID:** Removes a function from the database using its ID.
- **Delete Function by Name:** Deletes a function identified by its name.
- **Delete All Functions:** Clears all function entries from the system.

Utility Endpoint:

- **Check Requirements:** Evaluates which nodes meet the specified requirements of a given function, aiding in optimal deployment decisions.

These endpoints collectively support the management and allocation of resources and functions, ensuring efficient operation across a distributed computing landscape.

Figure 14. FastAPI visualization

6. Trustworthy-driven Policy Validation

6.1 Introduction and Objectives

The **Trustworthy-driven Policy Validation** component is a key part of the MANOLO architecture. It is designed to enforce policies related to **operational performance** (during experiment execution), **eco-efficiency**, and **trust** in AI workloads running on **distributed and heterogeneous infrastructures**.

All experiments in MANOLO are managed by the **Benchmarking component**, which is responsible for orchestrating their execution. Within this framework, the Policy Validation component ensures that AI models behave according to predefined policies. It doesn't apply corrective actions directly; instead, it monitors compliance, detects violations, and supports both real-time and retrospective analysis. During the execution of an experiment, it provides **live insights** to the MANOLO developer, across a Graphical User Interface, helping to identify deviations from expected behavior as they occur. MANOLO developers may implement by-hand recovery actions. Once the experiment is complete, the same component supports **post-mortem analysis**, offering a clearer understanding of system behavior and the effectiveness of the applied policies. By supporting both live monitoring and detailed after-action reviews, the Trustworthy-driven Policy Validation component helps improve operational visibility, supports continuous refinement, and reinforces the overall robustness of the MANOLO system.

6.2 Methodology and Implementation

The **Trustworthy-driven Policy Validation** component relies on a modular and programmable architecture that separates policy lifecycle management, metric evaluation, and monitoring integration. It is designed to be both developer-friendly and technically extensible, making it a key enabler for fine-grained, context-aware validation of AI workloads. At the center of this system is the **Policy Manager**, which governs the lifecycle of policies—registering, managing, and enforcing them—without executing remediation or recovery actions. Its role is to monitor compliance by evaluating system metrics, raise alerts when violations occur, and support **post-mortem diagnosis** for system improvement and trust reinforcement.

Policies are defined as **rules** that are continuously evaluated against **metrics collected from the infrastructure and workloads**. While the current implementation emphasizes three main categories—**performance**, **eco-efficiency**, and **trust**—the design is intentionally **extensible** to accommodate additional metrics specific to each experiment or AI service. For instance, performance metrics may include latency, throughput, inference time, or jitter. Eco-efficiency metrics can relate to energy use per task, resource utilization, or idle-time waste. Trust metrics might address model integrity, input data origin, or consistency in inference behavior. Crucially, developers can also define policies based on custom execution-related metrics—for example, confidence scores, drift indicators, or pipeline-specific variables. These metrics, whether standard or custom, can be injected into the system and used as **policy baselines**, enabling tailored and expressive monitoring strategies.

The **GUI** plays a critical role as the **main portal for MANOLO developers**. Through this web-based interface, developers can define, register, and manage policies, inspect violations in real-time, and review historical compliance data. The GUI provides visibility into the entire policy landscape—showing which policies are active, which have been violated, and the state of the system over time. This simplifies interaction with the

policy system and ensures that the developer or operator remains informed throughout the experiment lifecycle.

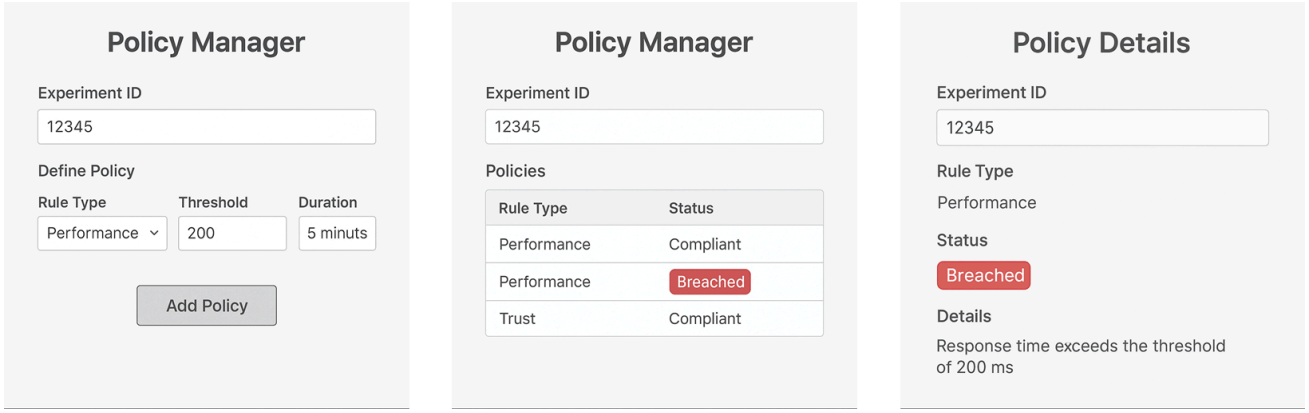


Figure 15. Dashboard mockup

Behind the GUI, the **PolicyEngineLibrary** provides the programmatic interface between the frontend and backend. This Python-based library handles policy submission, query operations, and violation tracking. It acts as a client to the backend infrastructure and simplifies the integration of policy logic into the experiment workflows. On the backend, the system extends the capabilities of **Thanos**, which serves a dual purpose: it stores both **the time-series metrics** collected from monitored systems and **the policy definitions and status data**. This ensures consistency between observed data and enforced rules, and supports historical analysis. The **Thanos Ruler** has been extended to support **dynamic and programmable evaluation of policies**, continuously checking incoming metric streams against the defined conditions.

For example, a developer might define a policy such as:

> *"If the average model confidence score drops below 0.75 over a sliding window of 100 inferences, and CPU usage exceeds 80%, raise a warning."*

Such policies are automatically evaluated by the backend, with alerts triggered and logged whenever the rule is violated. This evaluation process is **non-intrusive**—it monitors and reports without interfering with the experiment execution—ensuring clear separation of concerns and compatibility with other components like orchestration or benchmarking.

As illustrated in **Figure 16: Trustworthy-driven Policy Validation Sequence Diagram**, the architecture supports a **loop of continuous monitoring, real-time visibility, and historical traceability**, forming a programmable trust enforcement layer that aligns with MANOLO's goals of transparency, operational insight, and trustworthy experimentation.

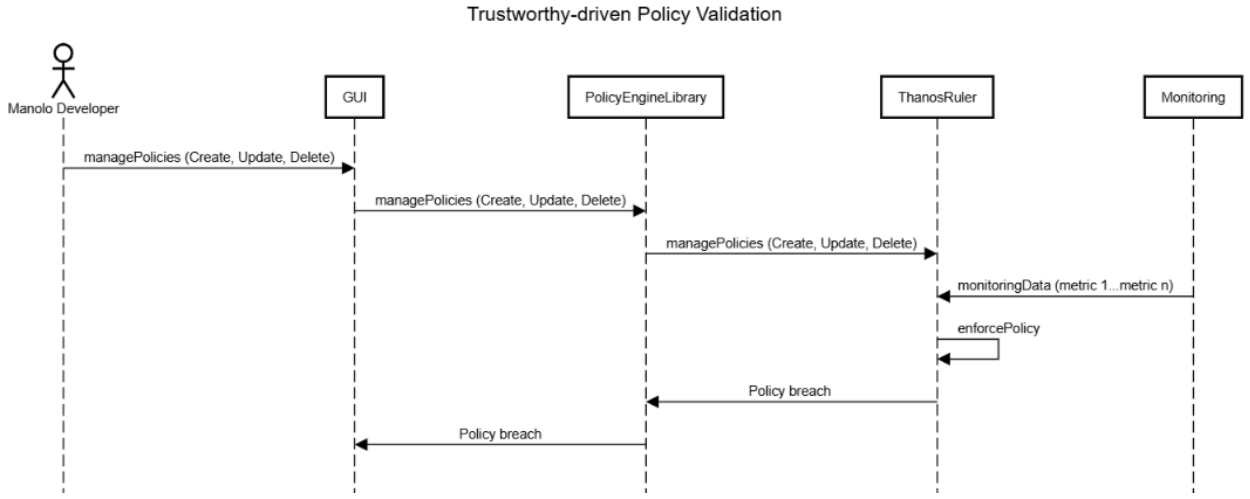


Figure 16. Trustworthy-driven Policy Validation Sequence Diagram

6.3 Experimental Results and Alignment

Initial **laboratory experiments** have been conducted to validate both the **conceptual soundness** and **technical integration** of the Trustworthy-driven Policy Validation component within the broader MANOLO architecture. These tests focused on evaluating how policies—defined over diverse and programmable metrics—can be enforced in near real-time without introducing significant overhead or latency.

The component has been integrated with several other system modules, including the **monitoring stack (based on Thanos)** and service orchestration layers, to ensure seamless data flow and policy evaluation continuity. Using controlled deployments and synthetic workloads, we tested:

- Real-time detection of violations (e.g., latency breaches, energy thresholds)
- Backend evaluation via the **extended Thanos Ruler**
- Visualization of alerts and system status through the developer frontend

These experiments demonstrated that the Policy Manager:

- Accurately detects violations based on programmable rules
- Operates with **minimal latency**, confirming its suitability for both development and operational phases
- Successfully integrates with other system components, confirming the **interoperability and modularity** of the MANOLO architecture

Furthermore, the experiments confirmed the value of **programmable metric injection**: by simulating a wide range of conditions and custom metrics (including AI model execution characteristics), we validated the flexibility of the system to adapt policy logic to diverse vertical requirements.

This component directly aligns with MANOLO's overarching goals of enabling **observability**, **trust**, and **runtime assurance** in cloud-edge AI environments. It strengthens the system's ability to enforce compliance through transparent and adaptable mechanisms—critical for ensuring accountability and robustness in AI-driven deployments.

7. Trustworthy & Infrastructure Oriented Model and Learning Allocation

7.1 Technical Requirements

This component has following technical requirements:

Table 5. Technical requirements for T4.4

Programming Language	• Python Version: 3.8 or newer.	
Required Libraries	• Fast API • Uvicorn • JSON • YAML	
Input Requirements	***User Request***	• Performance Requirements: May include response time, accuracy, power efficiency, and data privacy. (Optional) • Infrastructure • AI Workload
	For Training Tasks	• MANOLO Developer must provide the infrastructure information (nodes with available resources such as CPU, RAM, Storage, and GPU). (Must)

	For Inference Tasks	• If the MANOLO Developer is not providing a pre-trained model, the Manolo catalogue must contain the model with its:
		  ○ Estimated performance metrics.
		  ○ Minimal hardware and software requirements.
		• This component must receive the filtered topology (available infrastructure) from T4.2.

Dependencies for Optimal Resource Allocation	• The benchmarking database must be available to fetch the performance history of models. • Manolo's model catalogue should be up to date with all available models, estimated performance and their requirements. This component ensures that it provides the best and most suitable solution to the user only if all the above-required information is provided.	
Containerization (Optional for Future Integration)	• The component can be containerized using Docker for consistent deployment and scalable integration with other components. • It can be exposed and integrated with other components either via API calls or as a Dockerized service.	

7.2 Component Description

This component is responsible for identifying the best resources across the available infrastructure in response to user requests. This module supports two primary types of allocation scenarios:

- Inference
- Training Tasks, including both centralized training and Federated Learning (FL)

Moreover, in case of inference, and when there is more than one available model to execute a specific task, the component also optimizes the model selection.

The input request for this task is composed of two YAML files, one with the MANOLO Developer request including policy definition such as response time, accuracy, power efficiency, data privacy, etc.; as well as the subset of infrastructure nodes where to run the experiment. The second YAML file is the AI workload, where the kind of experiments to be executed are described, including the specific node where each one of the experiments will be executed, see Figure 4.

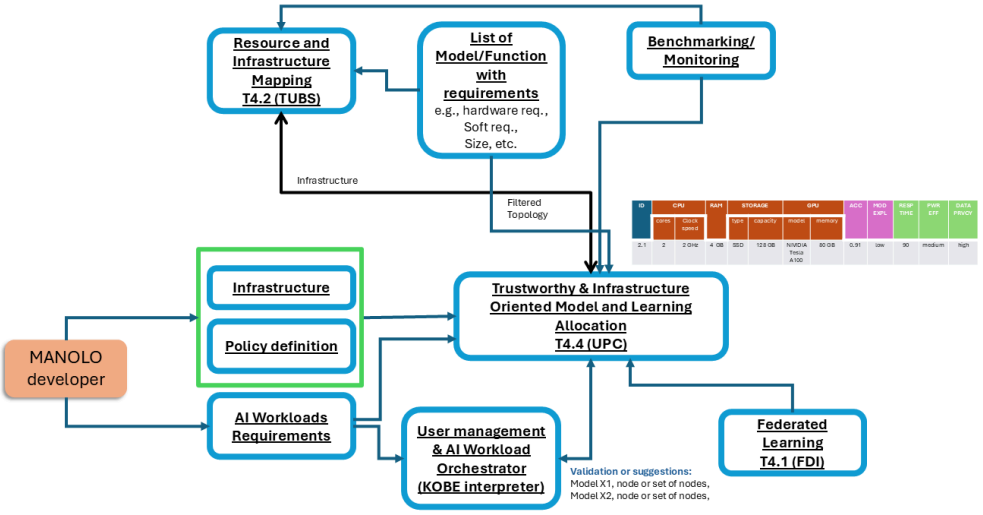


Figure 17. T4.4 Architecture

The YAML with the AI workload serves both as an input to the AI workload orchestrator (KOBE-WP5) and as an input to T4.4, which for each AI workload checks against T4.2 if the specified node is available, and if it has enough resources, in terms of hardware and software, for executing the experiment.

To check the hardware and software availability of the requested node, T4.4 forwards the node request from the AI workload orchestrator to T4.2, which answers if the node is available or not, as well as all the hardware and software characteristics of the node in case of being available. This is called filtered topology.

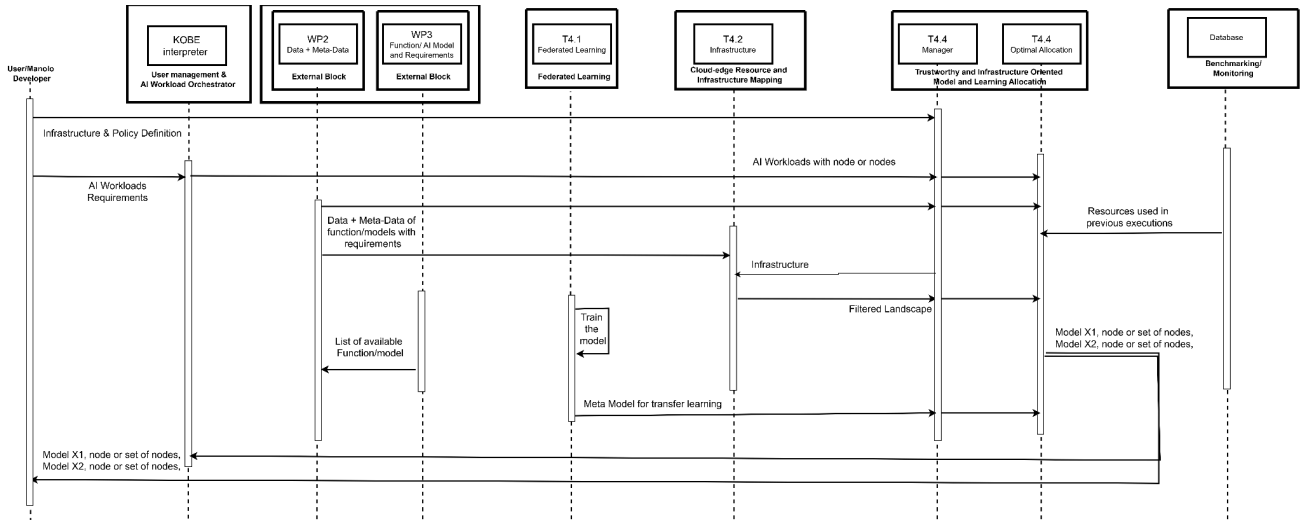


Figure 18. T4.4 Overview

MANOLO's first version

In a first version of MANOLO, M18, if the answer provided by T4.2 is confirming the hardware and software availability, then T4.4 checks the Benchmarking database if the provided node can meet the user requirements. After this second filtering a confirmation is sent to the AI workload orchestrator (KOBE) in case of node availability. When the node is not available T4.4 may provide a suggested list of available nodes following the same process described below for the second version of MANOLO.

MANOLO second version

For a second version of MANOLO, the AI workload orchestrator (KOBE) will not provide the specific node where the experiment will be executed, but T4.4 will run an optimization process, providing to the AI workload orchestrator the list of nodes with enough availability, ranked according to the user requirements.

This process of optimization will follow these key steps: after receiving the filtered topology from T4.2, T4.4 first queries the benchmarking/monitoring database to check for previous executions of the same task. If an earlier execution meets or exceeds the requested performance metrics, the system retrieves the associated resource footprint (CPU, RAM, storage, GPU, etc.) used in that execution.

The resource demands extracted from benchmarking/monitoring are considered the minimum or baseline requirements for fulfilling the current task under similar constraints. Finally, the component evaluates the current state of the available infrastructure to identify nodes that can satisfy or exceed the identified requirements. The matching is initially based on parameters such as CPU (cores and clock speed), RAM, storage (capacity and type), GPU (model and memory), etc. See Figure 18.

When there are no previous executions, the component will utilize information from WP3 estimation metrics, such as number of FLOPs or Weighted MAC Count to estimate time response and power consumption.

This approach identifies the node or sets of nodes that can fulfil the user request. Furthermore, an intelligent ranking mechanism will be developed to prioritize nodes and execution strategies based on performance and user preference. In case of inference the component also optimizes the selection of the model according to the hardware constraints and the user requirements and preferences.

7.3 Workflow

The Resource Allocation task (T4.4) serves as the core decision-making module within WP4. It operates by integrating inputs from various tasks and computing optimal resource-to-task mappings based on user-defined requirements (see workflow in Figure 19):

Model extraction: Based on the YAML with the AI workload, where the model to be executed is specified, T4.4 extracts the suitable list of models performing the same task that meet the user requirements from the Data Tier (Central Storage in Figure 19). Each extracted model is then evaluated against the benchmarking/monitoring database to identify any previous executions.

Benchmark/Monitoring: Task T4.4 queries the benchmarking/monitoring database to identify any historical executions of the same or similar tasks. In Figure 21, we can see an example of Benchmarking and Monitoring metrics used by T4.4; with the categorization of these metrics visualized in different colours. If matching entries are found that meet or exceed the requested performance criteria, the used/consumed resources associated with those executions (e.g., CPU, RAM, storage, GPU) are extracted. If no previous executions are found the task may use estimated values coming from WP3.

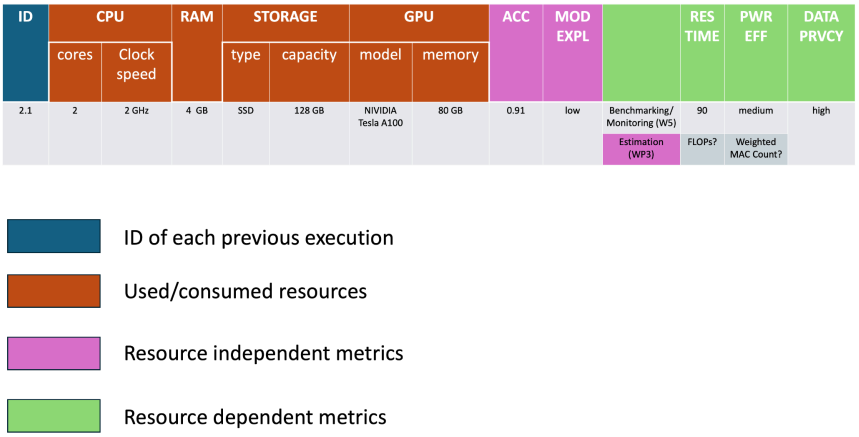


Figure 19. Input from Benchmarking/Monitoring and WP3 metrics

Resource Mapping: T4.4 uses the extracted models and attempts to map them on matched nodes or a set of nodes from the available infrastructure provided by T4.2. This step involves evaluating whether current node capabilities satisfy the requirements derived from benchmarking/monitoring.

Evaluation and Ranking: In scenarios where multiple valid combinations of models and nodes are identified, an evaluation score is calculated for each candidate pair. The scoring mechanism considers the user's performance preferences and the degree to which the available resources match the task requirements.

Optimal Resource Selection: Based on the evaluation scores, candidate mappings are ranked in descending order. The top-ranked node or set of nodes is then selected as the optimal deployment target for the task. In case of inference the top-ranked are pairs of model-node combinations.

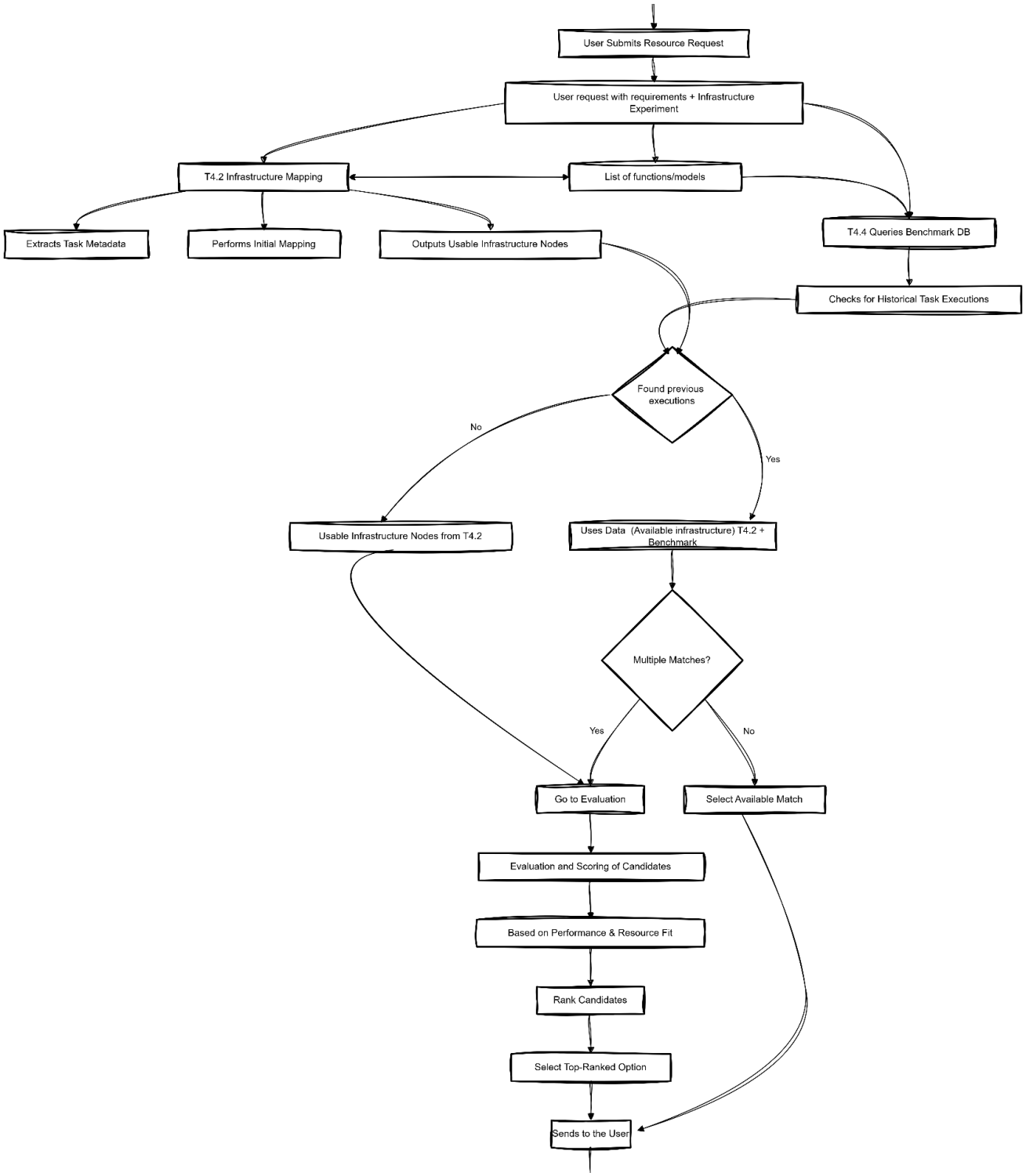


Figure 20. T4.4 Workflow

7.4 Implementation

Task 4.4 is divided into multiple subcomponents (see Figure 20), each responsible for a distinct functionality in the workflow. The implementation details of each component are described below.

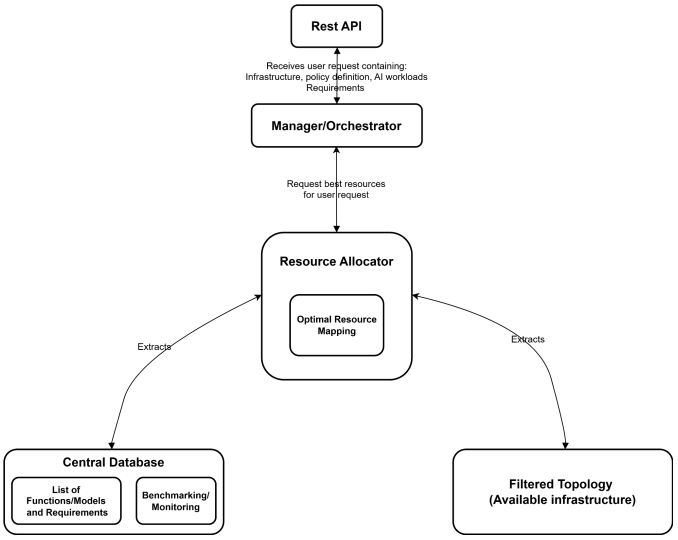


Figure 21. Main T4.4 subcomponents

REST API: Component-FastAPI (Python):

This component exposes the resource allocation service to other modules or users. It acts as the entry point for user requests.

Input:

- Format: YAML

Contents:

- task_name: Identifier for the task
- task_type: function, inference, or training (includes FL)
- Users policy definition: e.g., accuracy, latency, privacy, power efficiency, etc.
- Infrastructure
- AI workload

Output:

- Format: JSON/YAML (TBD)

Contents:

- Mapping of each requested task with best suitable node
- Reason of failure (e.g., not enough resource)

Manager/Orchestrator: Component (Python):

This component orchestrates the resource allocation. At this moment, it just translates the YAML file into a python dictionary and sends the request for resource allocation to Resource Allocator.

Input:

- From REST API (parsed YAML as Python dict)

Resource allocator (Component: Python):

This component performs the main functionality of resource allocation. It processes user input, fetches required data from internal modules and services.

- Extracts and filters models/functions that meet user performance requirements.
- Queries benchmarking records to get historical executions and resource usage.
- Retrieves filtered infrastructure topology from Task T4.2 (e.g., usable node list).
- Filters the combinations (function/model with node/nodes) that meet or exceed the user requirements.

For further optimization, an algorithm to calculate the evaluation score will be developed that will prioritize the user requirements and will rank the filtered node in descending order.

Input:

- User request (application file)

Extracts internally:

- Filtered functions/models
- Benchmark entries for each task
- Filtered nodes

Output:

- Format: JSON/YAML (TBD)

Contents:

- Mapping of each requested task with best suitable node
- Reason of failure (e.g., not enough resource)

The snapshots of the output after executing this component for a specific user request are shown in Figure 22 left; alongside is the return output resource.json file, Figure 22 right with the mapped result for each task with the best suitable nodes.

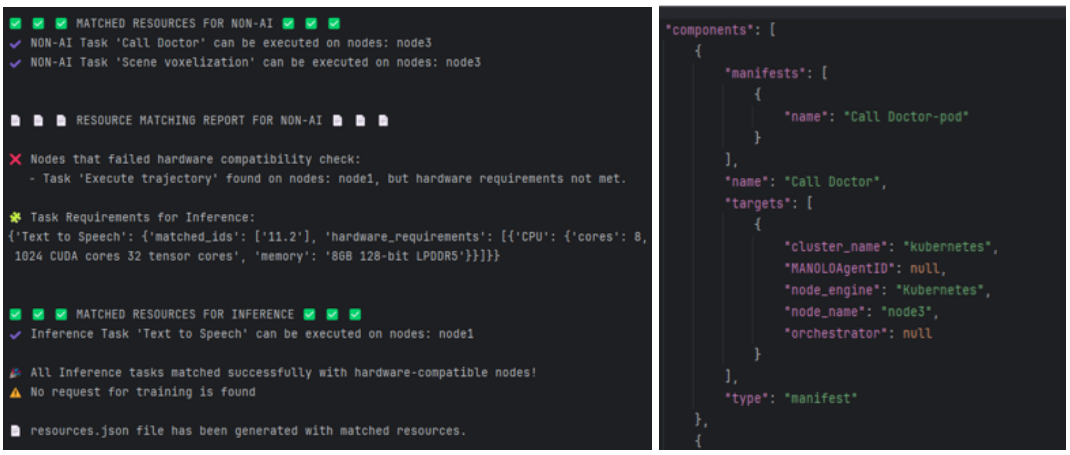


Figure 22. Snapshots with the output of the component.

8. Use Case Alignment

8.1 Overview of Use Cases

MANOLO validates its framework through three representative pilot scenarios that stress complementary aspects of the cloud-edge continuum, resource optimisation, and trustworthiness.

Use Case 1: Robotics in Healthcare & Industry

• Context: PAL Robotics' TIAGo platform interprets spoken commands and non-verbal cues to execute manipulation tasks.

• Variants: UC 1.1 caregiver support in hospital wards; UC 1.2 collaborative assembly on factory floors.

• MANOLO focus: Real-time scheduling of multi-modal perception and grasp-planning models across robot, on-prem edge server, and public cloud while enforcing sub-100 ms latency, safety "emergency-stop" rules, and GDPR-compliant data handling.

Use Case 2: Mobile Computer Vision

• Context: a white-label mobile app for telecom providers that auto-organises encrypted photo albums via image classification, face detection, and user-custom model training.

• Key needs: efficient model compression, on-device inference vs. cloud fine-tuning, and user-facing explainability ("Explain AI").

• MANOLO focus: dynamic placement of common and user-specific models, privacy-by-design data flows, and scalable resource allocation to support hundreds-of-thousands of concurrent users.

Use Case 3: Auditory Stimulation for Memory Consolidation

• Context: a home-use closed-loop system that employs EEG-driven AI to deliver precisely timed auditory cues during slow-wave sleep, aiming to improve memory in MCI/dementia patients.

• Key needs: low-latency edge inference on wearable/headband data, efficient retraining of detection models, and secure handling of sensitive physiological signals.

• MANOLO focus: optimize training and deployment pipelines across local gateways and cloud back-ends, minimising energy use while guaranteeing data protection and continuous operation. Together these pilots cover real-time robotics, large-scale consumer AI, and medical IoT therapy, providing a comprehensive test-bed to demonstrate MANOLO's capabilities in trustworthy, resource-aware AI deployment across the cloud-edge continuum.

8.2 Use Case 1: Robotics in Healthcare & Industry

The input from UC1 for the topology and AI tasks is aligned with the work in WP4.

• ***Three-tier edge-cloud layout***

- node1 TIAGo mobile robot (on-board compute).

- node2 On-prem edge server inside PAL facilities.

- node3 Public cloud VM hosted in Spain (VULTR).

• *node1: TIAGo robot*

- Sensors RGB-D, microphone, LiDAR → multi-modal perception.
- Actuators mobile base, torso, head, 7-DoF arm, speaker.
- HW 8-core CPU @2.3 GHz, 32 GB RAM, 8 GB Ampere GPU (7-25 W).
- Local action set motion, grasp-planning, speech I/O, emergency calls.
- Network 100 ms / 30 Mbps to edge; 300 ms / 10 Mbps to cloud.
- Security ECDSA auth; EU Trustworthy-AI tag.

• *node2: Edge server (same building)*

- HW 16 cores (3.2–5.2 GHz), 32 GB RAM, RTX 3080 Ti (12 GB).
- Network 100 ms to robot, 10 ms / 500 Mbps to cloud.
- Actions heavy vision modules, speech-to-text, VLA grasping.
- Full data-encryption; Trustworthy-AI tag.

• *node3: Cloud server*

- HW 12 cores @3.5 GHz, 120 GB RAM, Tesla A100 (80 GB).
- Cost 2.4 USD / h, 10 TB monthly traffic cap.
- Network 10 ms to edge, 300 ms to robot.
- Actions model fine-tuning (LoRA), TTS, large-scale perception, alerting.
- Encrypted storage, compliance tag.

• *Cross-node characteristics*

- Latency tiers guide real-time vs. batch placement.
- Bandwidth asymmetry (robot ↔ cloud 5 Mbps) discourages raw-video upload.
- Compliance tags common to all nodes simplify policy reasoning.

Contribution to WP4

• *Machine-readable map*

- The JSON file is ingested by T4.2 to populate the Landscape DB (compute, memory, accelerator, energy, cost, network, security).

• *Unified descriptors*

- “type”, “hardware”, “network”, “actions” fields are converted to WP4 abstraction parameters, enabling heterogeneous resources to be compared uniformly.

• *Capability catalogue*

- Action lists tell T4.2 which AI functions each node can host natively; this speeds up feasibility checks and shortens deployment time.

• *Constraint repository*

- Latency/bandwidth values, power budget of the robot GPU, and hourly cloud cost are stored as hard constraints for the Allocation Manager (T4.4).

• *Compliance baseline*

- Security and "AI HLEG" annotations are registered so that T4.3 Policy Manager can later enforce GDPR-grade data localisation (e.g., patient data must stay on node1/2).

• *Dynamic updates*

- T4.2 serves the topology to T4.4 in every validation loop ("Hello-World" flow) so that scheduling decisions always reflect the latest resource status.

In short, the UC1 topology file encapsulates all hardware, network, capability and policy information required by WP4. Task 4.2 uses it to build the landscape database, while Task 4.4 leverages the same structured data to make latency-aware, policy-compliant placement decisions for the healthcare and industrial robotics scenarios.

8.3 Use Case 2: Mobile Computer Vision

The input from UC2 for the topology and AI tasks is aligned with the work in WP4.

Three-layer mobile-edge-cloud layout

• *node1: End-user mid-range mobile phone (e.g. Samsung A55)*
- Sensors: 4 rear + 1 front RGB cameras (50 MP → 5 MP) for photo capture.
- HW: Exynos 1480, 8 × A78/A55 cores @ 2.75 GHz, Xclipse 530 GPU, 8 GB LPDDR4x, 128 GB UFS 2.2.
- Energy: 5000 mAh battery, passive cooling → strict power/thermal limits.
- Network: 802.11n Wi-Fi, 12 ms RTT, 144 Mbps peak.
- SW: Android 14, Vulkan 1.3 acceleration.
- Actions already deployable on-device: "Classify photos", "Face detection", "Few-shot custom training", "Explain AI", etc.
- Security: AES-encrypted storage, WPA2-PSK link.

• *node2: End-user edge server*
- Location: User premises.
- HW: AMD Ryzen 7 1800X (8 c / 16 t), 48 GB DDR4, RTX 3060 (12 GB), NVMe 2 TB.
- Network: Gigabit Ethernet to phone, 2 ms RTT, 1 Gbps.
- SW: Windows 10, CUDA 11.2 + cuDNN 8.1, FlashAttention.
- Actions supported: all inference functions plus light training / model adaptation.
- Security: TLS 1.0–1.3, full-disk encryption.

• *node3: Cloud server)*

- Location: Premises of MLaaS provider.
- HW: AMD EPYC 9654 (96 c / 192 t), 256 GB RAM, 2 × NVIDIA A16 boards (8 GPUs total, 64 GB each), NVMe 1 TB.
- High-end compute & memory for large-scale training, model compression, batch analytics.
- Network: 1 Gbps Ethernet backhaul to edge; sub-5 ms RTT expected (exact value truncated).
- Cooling & PSU info included for energy modelling; always-on power.
- Security: encrypted storage, datacentre-grade controls.

How this topology supports WP4, especially Task 4.2 (Landscape Manager)

• *Unified resource inventory*
- JSON fields ("CPU.bits", "Accelerators.GPU_Board1.compute_capability", "Battery.capacity", etc.) are converted into WP4's standard descriptor schema, populating the Landscape-Manager's database.

• *Capability catalogue*
- "actions_supported" per node lets T4.2 know which AI functions (classification, custom training, explainability) are already deployable, easing feasibility checks for new experiments.

• *Constraint repository*
- Battery level, passive cooling limits, Wi-Fi vs. Ethernet latency and bandwidth, and encryption standards are stored as constraints that the Allocation-Manager (T4.4) must respect.

• *Dynamic cost/performance modelling*
- High-end GPUs and massive RAM on node3 allow T4.4 to decide when to spawn large retraining jobs or model-compression pipelines in the cloud, using T4.2's up-to-date bandwidth and latency data to avoid congestion.

• *Compliance tagging*
- Security annotations (TLS versions, AES, WPA2) are stored by T4.2 and exposed to the Policy-Manager (T4.3), ensuring that photo data never leaves encrypted channels and that GDPR-by-design rules are enforced.

• *Re-usable structure*
- The same JSON layout that feeds T4.2 is reused by T4.4 during every validation loop, enabling seamless resource mapping and policy-aware allocation for the Mobile Computer Vision use case.

8.4 Use case 3: Auditory Stimulation for Memory Consolidation

The input from UC3 for the topology and AI tasks is aligned with the work in WP4.

Three-level wearable-edge-cloud architecture
- node1 EEG headband worn by patient (embedded compute).
- node2 Neuromorphic Chip (neuromorphic processor ADELIA).
- node3 Subject tablet at home (edge).
- node4 On-prem edge server inside hospital ward.
- node5 AWS cloud instance (EU-Ireland region).

• *node1: EEG headband*
- HW: 4-core CPU @ 2.3 GHz, 8 GB RAM, 1 TB SSD (for long-term sleep logs).
- Network: 50 ms / 2 Gbps to edge; 200 ms to cloud.
- Role: Real-time EEG acquisition, sleep-stage detection, cue triggering.
- Security: On-device encryption; EU Trustworthy-AI tag.

• *node2: Neuromorphic Chip*
- HW: Neuromorphic Processor ADELIA.
- Network: 50 ms / 2 Gbps to edge; 200 ms to cloud.
- Role: On-prem computing device for EEG processing and stimulation.
- Security: On-device encryption; EU Trustworthy-AI tag.

• *node3: Subject Tablet*
- Location: inside the patient rooms.
- HW: *Microsoft Surface*.
- Network: 50 ms / 2 Gbps to edge; 200 ms to cloud.
- Role: system configuration tool, create reports, simple computations.
- Security: on-device encryption; EU Trustworthy-AI tag.

• *node4: Hospital edge server*
- Location: next to patient rooms (Getafe University Hospital).
- HW: 8-core CPU, 32 GB RAM, NVIDIA Tesla T4 (16 GB).
- Network: symmetrical 50 ms to headband, 200 ms to cloud.
- Role: heavier signal processing, model adaptation, local data vault.
- Security: encrypted storage, same compliance tag.

• *node5: Cloud server (AWS)*
- HW: 8-core CPU, 32 GB RAM, Tesla T4 (16 GB).
- Network: ≈100 ms to both headband and edge.
- Role: batch model training, analytics across multiple patients, archive.
- Security: full encryption; GDPR/AI-Act alignment via tag.

• *Cross-node links*
- Latency tiers (50 ms ↔ 100 ms ↔ 200 ms) guide what can run where in the closed loop.
- High bandwidth (2 Gbps–4 Gbps) allows periodic bulk transfers but not in-loop streaming.
- Uniform “AI HLEG” compliance tag simplifies policy enforcement.

Contribution to WP4

• *Machine-readable inventory*
- All CPU/GPU, memory, storage, network and security fields are parsed into WP4’s Landscape database.

• *Capability catalogue*
- Action list (sleep-stage detection, cue delivery, model updates) is stored per node so T4.2 can answer “Can this function run here?” during feasibility checks.

• *Constraint repository*
- Latency ceilings (< 100 ms for effective closed loop) and encryption requirements become hard constraints that the Allocation Manager (T4.4) must respect.

• *Privacy & compliance baseline*
- Medical-grade data encryption and AI-HLEG tags are surfaced to the Policy Manager (T4.3) for GDPR-compliant placement (e.g., raw EEG never leaves node1/2).

• Energy & locality awareness
- Knowing that node1 is wearable with limited compute, T4.4 (using data held by T4.2) can off-load heavy inference to node2 while still meeting 50 ms round-trip.

• *Reusable JSON structure*
- The same topology file flows from T4.2 (resource mapping) to T4.4 (allocation) within MANOLO's "Hello-World" and full pilot deployments, ensuring consistent, policy-aware scheduling for the auditory-stimulation use case.

9. Conclusions and Next Steps

This deliverable completely describes the first version in M18 of the Cloud-edge Continuum AI Model Function Allocation module in MANOLO. This module has different functionalities which can be summarized as: it must select the most suitable resources across the cloud-edge continuum, optimizing this selection depending on various metrics such as latency, throughput, energy efficiency, etc.; both for training and inference phases. If requested, Federated Learning must also be applied. And finally, the system must also check the policy violation.

The main progress in this MANOLO component has been the agreement on the main internal WP4 architecture, including two workflows described in 3.2, as well as the integration of it in the whole MANOLO architecture. WP4 is part of the MANOLO controller described in the MANOLO architecture, and for this reason also a Controller API will be developed in WP4. Through this Controller API the MANOLO developer can interact with the MANOLO framework. In the case of WP4, the controller API will receive requests for executing experiments (AI workload requirement YAML file), and WP4 will be in charge of validating the resource availability for executing this experiment. In addition, the Controller API will handle all policy-related management actions (create, delete, etc).

In this first version, Federated Learning has been completely deployed and experimented in one of the use cases, showing an improvement in terms of training time and energy consumption when FL is applied. This first implementation and validation of FL has been carried out in an isolated manner due to the special characteristics of this task. However, in the WP4 architecture, FL-based resource allocation is already foreseen and supported.

On the other hand, mapping (T4.2) and resource allocation (T4.4) have been working in coordination, and in this first approach, the tasks work together to check the availability of the node proposed by the AI Workload Orchestrator to execute each one of the experiments. In this simple approach T4.2 checks the hardware and software availability according to model requirements; and T4.4 checks the possible availability taking into account the user requirements in terms of accuracy, latency, etc by using data from previous executions stored in the Benchmarking database. Meanwhile, policy compliance (T4.3) is checked at runtime to detect violations and generate alerts for the MANOLO developer.

For the second part of the project, new and optimized resource allocation mechanisms will be developed, both for training and for inference, giving the user a ranged list of possible nodes and possible models where to execute the experiment. Also, when there is no availability, new recommendations will be provided. Examples of these possible recommendations may be: enlarge the set of nodes in the infrastructure, compress the algorithm, or apply Federated Learning.

Another field of research will be to specifically define the set of policies that MANOLO can check for possible violation based on the use case requests, as well as the possible list of provided alerts.

Finally, a new field of research will be developed in the framework of MANOLO related to the resource trustworthiness, trying to parametrize each MANOLO node with a trustworthiness score.

MANOLO

CLOUD-EDGE EFFICIENT & TRUSTWORTHY AI

GA 101135782

Partners

CeADAR Ireland's Centre for AI

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Atos

EVIDEN

Technische Universität Braunschweig

DEMOKRITOS

4DOT

Fraunhofer IIS

Inria

université PARIS-SACLAY

ARCADA

KU LEUVEN

LAUREA

PAL ROBOTICS

Bitbrain

arx.net MOBILE INNOVATIONS

Q-PLAN INTERNATIONAL

eit Digital

@ manolo-project

Funded by the European Union