AIoTwin

Twinning action for spreading excellence in Artificial Intelligence of Things

DELIVERABLE D1.1

REPORT ON USE CASES, REQUIREMENTS, AND ARCHITECTURE

Funded by the European Union

Project number: 101079214
Project name: Twinning action for spreading excellence in Artificial Intelligence of Things
Project acronym: AIoTwin
Call: HORIZON-WIDERA-2021-ACCESS-03
Topic: HORIZON-WIDERA-2021-ACCESS-03-01
Type of action: HORIZON Coordination and Support Actions
Granting authority: European Research Executive Agency

DOCUMENT CONTROL	
Deliverable No.	**D1.1**
Title	Report on Use Cases, Requirements, and Architecture
Lead Editor	Mario Kušek
Type	Report
Dissemination Level	PU
Work Package	WP1
Due Date	31.12.2023

AUTHOR(S)

Name	Partner	e-mail
Mario Kušek	UNIZG-FER	mario.kusek@fer.hr
Ivana Podnar Žarko	UNIZG-FER	ivana.podnar@fer.hr
Dora Kreković	UNIZG-FER	dora.krekovic@fer.hr
Ivan Čilić	UNIZG-FER	ivan.cilic@fer.hr
Ivan Kralj	UNIZG-FER	ivan.kralj@fer.hr
Mislav Has	UNIZG-FER	mislav.has@fer.hr
Duc-Manh Nguyen	TUB	duc.manh.nguyen@tu-berlin.de

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.01	03-05-2023	Mario Kušek	Initial template and structure
0.02	01-07-2023	Mario Kušek	Initial idea about middleware requirements and architecture
0.03	30-09-2023	Mario Kušek	Refinement of initial idea
0.04	01-12-2023	Dora Kreković, Ivan Čilić, Ivan Kralj, Mislav Has	Introduced parts of research on SOTA and use cases
0.05	08-01-2024	Mario Kušek, Dora Kreković, Ivan Čilić, Ivan Kralj, Mislav Has	Initial architecture and use cases

0.06	11-01-2024	All	Comments on initial architecture and use cases
0.07	17-01-2024	Mario Kušek, Dora Kreković, Ivan Čilić, Ivan Kralj, Mislav Has	Initial requirements and connection to use cases, specific architecture for each use cases, added data that will be used in use cases
0.08	18-01-2024	All	Comments on architecture, use cases and data
0.09	31-01-2024	Dora Kreković, Ivan Čilić, Ivan Kralj, Mislav Has	Improving existing content
0.10	27-02-2024	Ivana Podnar Žarko	Internal review
0.11	01-03-2024	Duc-Manh Nguyen	Added dataset description used by TU Berlin
0.12	08-03-2024	Dora Kreković, Ivan Čilić, Ivan Kralj, Mislav Has	Content finalization based on internal review
1.00	11-03-2024	Mario Kušek	Final version for submission

Table of Contents

Executive Summary

This deliverable presents the use cases, system requirements and architecture of the AIoTwin data-driven orchestration middleware. It contains an analysis of the current state of the art in the field of Artificial Intelligence of Things (AIoT), focusing on three specific research areas: orchestration in the edge-to-cloud continuum, federated and decentralised learning, and robust energy-efficient IoT. Following the state of the art, Section 3 describes relevant use cases intended for testing the developed middleware, associated mechanisms, and algorithms. Two general use cases are foreseen, the first one in the context of smart city for traffic management and the second one on smart agriculture and continuous monitoring of plants and their environment. In the same section, the available datasets for each use case are listed. Section 4 identifies the middleware requirements and lists specific requirements for each use case. The next section defines the AIoTwin middleware architecture. It starts with a general architecture for the orchestration of machine learning (ML) pipelines with a description of its generic components. It then refines the general architecture to define a more specific architecture for three specific AIoT problems: 1) adaptive orchestration of federated learning (FL) pipelines, 2) QoS-aware load balancing for inference services in the edge-to-cloud continuum, and 3) inference for efficient and energy-aware communication.

1 Introduction

1.1 Deliverable context and description

Work Package 1 (WP1) comprises the tasks and activities of the AIoTwin Joint Research Project, which incorporates the contributions and expertise of all AIoTwin partners. In this joint research, we are focussing on the following goals:

- Develop a data-driven orchestration middleware for energy-efficient IoT supporting ML workflows.
- Investigate the mechanisms for orchestration of containers across the edge-to-cloud continuum.
- Run and test the middleware on the available infrastructure of all consortium partners.

This research is related to the following three AIoTwin research domains:

- Orchestration in the edge-to-cloud continuum,
- Federated and decentralised learning and
- Robust and energy efficient IoT.

In this work package, we will develop an edge orchestration middleware that proposes the placement of ML models in the edge-to-cloud continuum, taking into account data streams originating from many heterogeneous IoT devices, and explore the energy efficiency of edge orchestration deployments supporting ML workflows. In addition to optimising the placement of containers running AI/ML algorithms considering the available resources, QoS constraints and overall energy consumption, the focus is also on managing ML workflows and data routing in the edge-to-cloud continuum.

This middleware will be the result of joint research activities within WP1; however, it will be driven by research questions posed and investigated mainly by PhD students from UNIZG-FER participating in the AIoTwin project which are also relevant to their dissertation topics. The developed middleware will be published as open source in the AIoTwin GitHub repository (https://github.com/aiotwin) and will consist of libraries and components that can be used in different use cases in the field of smart city and smart agriculture which are introduced in Section 3.

1.2 Deliverable outline

This document is organised as follows. Section 2 analyses the current state of the art in the field of Artificial Intelligence of Things (AIoT). Section 3 identifies and discusses relevant use cases and available datasets. After describing the use cases, Section 4 identifies and describes the requirements for the middleware. Some requirements are general in nature, while others are specific to a particular use case. Section 5 presents the general AIoTwin architecture and introduces specific architectures for each of the envisaged use cases, which are defined as refinements of the general architecture. Section 6 concludes the deliverable and Section 7 contains a list of acronyms used. A list of figures, tables and references can be found at the end of the document.

2 State-of-the-art analysis

Internet of Things (IoT) solutions are inherently distributed and decentralised as they connect heterogeneous devices to the Internet via various communication technologies and enable the transmission of sensed data from the environment to nearby devices at the edge of the network and the forwarding of data to remote servers in the cloud in a so-called computing or edge-to-cloud continuum. In addition, IoT solutions facilitate actuation at the edge of the network by forwarding decisions and signals from the cloud or from nearby edge servers to devices or machines (e.g. robots) with actuation capabilities.

AI algorithms and concepts are being integrated into the edge-to-cloud continuum to analyse and learn based on data continuously generated by many devices to make decisions, predict future activities and autonomously operate devices at the edge of the network. Therefore, a new concept known as Artificial Intelligence of Things (AIoT) has emerged [1]. It brings artificial intelligence into the physical environment and requires novel machine learning (ML) approaches that are adapted to edge resources and the characteristics of the edge-to-cloud continuum while being able to process data that is continuously generated in the real physical environment. AIoT breaks down the boundaries between the physical and digital worlds and opens up new possibilities for device intelligence and autonomy in the physical environment, which is important for a wide range of applications, from smart factories and agriculture to the smart cities of the future.

AIoT also poses particular challenges for the field of artificial intelligence. A distributed and heterogeneous environment with limited resources in terms of available computing power and energy requires the use of efficient algorithms that are adapted to the distributed environment and utilise appropriate service orchestration in the continuum [2]. As data processing takes place in real time, machine and deep learning algorithms need to be adapted, bearing in mind that data streams from IoT devices are often incomplete, prone to errors, and unlabelled. In addition, AIoT systems must adhere to strict privacy and security requirements to protect sensitive user data and ensure the integrity of the devices and physical environment. Also note that energy efficiency is an important requirement in AIoT environments that utilise edge devices. Below we provide a brief overview of the latest solutions for edge orchestration, federated and decentralised learning and energy-efficient IoT that are relevant to the middleware to be developed and implemented as part of the AIoTwin project.

2.1 Orchestration in the edge-to-cloud continuum

Most of today's IoT data traffic is transmitted over the Internet towards remote cloud servers for processing or storage. However, such an architectural approach gradually overloads the network, lengthening the overall processing cycle for IoT services and reducing responsiveness to events detected in local intelligent environments. The concept of **Edge-to-Cloud Continuum** (ECC) has been developed to reverse this trend and significantly reduce the traffic generated towards the cloud by enabling the processing of IoT data closer to the data sources.

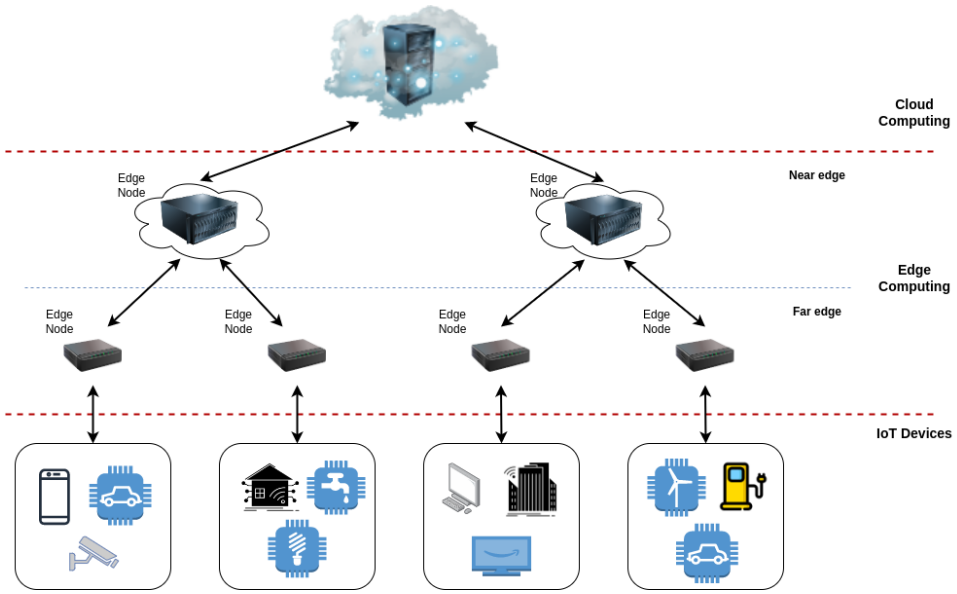

Figure 1. Abstract view of the edge-to-cloud continuum

In the ECC, the devices are organised hierarchically in layers, as shown in Figure 1. The top layer is a cloud computing layer with devices placed in the cloud. The subsequent layer between the cloud and IoT devices which are placed at the bottom of the continuum is the edge computing layer hosting edge nodes. It is divided into two parts: The upper part is called the **near edge**, as it is located close to the cloud, while the lower part is called the **far edge**. The bottom layer is the IoT device layer, which mainly contains resource-constrained IoT devices, while **edge nodes** are also resource-constrained compared to the cloud. A more detailed description of the units in each layer of the continuum can be found in [3]. It is important to note that each layer offloads the upper layer and performs a part of continuum functions. In addition, the nodes within the same layer are interconnected to share the processing load and optimise the placement of the deployed services. In this way, the processing and storage capacities are brought closer to the end IoT devices, which offers the possibility of achieving the following crucial objectives of the IoT concept [4]: reduced overall network traffic, improved responsiveness and shorter processing cycles, improved security with privacy control and lower operating costs.

In an ECC environment where end devices and edge nodes are constantly changing their state and location, manual management of services becomes complex and should be avoided by using an automated approach enabled by a suitable **general purpose orchestration tool**, e.g., Kubernetes, KubeEdge, K3s or ioFog. Thus, services running on the edge nodes should be orchestrated automatically to ensure their high availability.

Service orchestration in the ECC implies scheduling, deploying and managing services based on a specific scheduling policy within a dynamic and unstable execution environment. The challenge of implementing efficient service orchestration has been analysed mainly in the scope of cloud administration before the emergence of edge computing [5]. Thus, different approaches and orchestration systems already exist, but they must be adapted to become more suitable for the edge computing environment. Current

implementations of orchestration tools typically involve many features and capabilities to ensure system scalability and reliability across cloud environments. However, such capabilities make them resource-demanding and often too heavy for IoT devices and edge nodes. Furthermore, as many features needed for the cloud infrastructure are not critical in edge computing use cases, their number could be reduced to achieve optimized and lightweight versions of existing orchestration tools. Such versions should primarily include features necessary to execute efficient orchestration in the edge computing environment, where the emphasis is on specific performance targets rather than goals critical for cloud-based systems [6].

Edge services handling IoT data must be autonomous, stateless, and portable to ensure short migrations and high availability. Container virtualization is preferred for easy migration and reduced startup time. Thus, efficient service orchestration with containerized IoT components is essential for ECC benefits. A relevant survey examining containerization and scheduling of edge services is detailed in [7]. The authors offer insights into Kubernetes and Docker Swarm schedulers, along with algorithms utilized for efficiently scheduling container-based services in edge computing contexts. Additionally, in [8], the authors present a survey on edge orchestration, particularly focusing on container orchestration tools. A similar investigation is outlined in [9], which evaluates the performance implications of using Docker containers for IoT applications in fog computing setups. The authors propose a framework for deploying IoT applications at the edge using Docker Swarm. Furthermore, a comprehensive exploration of fog/edge orchestration challenges is documented in [5], where the authors provide an overview of the state-of-the-art of service orchestration and discuss technologies aimed at addressing major orchestration hurdles.

2.2 Federated and decentralised learning

The traditional ML pipeline assumes that data is collected and trained in a centralised cloud location. However, this approach comes with privacy concerns, as the data is available in raw format to both the services executing the training and the cloud providers. Also, the cost of sending the data to the cloud and the cost of storing and processing this data calls into question the viability of these solutions.

2.2.1 Federated learning

Federated learning (FL) is proposed to address the aforementioned privacy and network-related challenges by allowing the model to be trained on a federation of participating devices without the need to store the data centrally [10][11]. The collected data remains on the IoT devices (in FL they are referred to as *clients*) and only model updates are sent to the central coordination server (*aggregator*). The models obtained are then aggregated and sent back to the FL clients. This approach ensures data privacy and saves network bandwidth as the raw data does not leave the source devices, but only in situations where the models are smaller compared to the raw data. In addition, by distributing the training task across different devices, the computing and storage costs of the central server are significantly reduced, while the storage capacity and processing capabilities on the clients must be increased.

2.2.2 Decentralised learning

Training ML models on data distributed across individual devices, known as **peer-to-peer (P2P) learning**, presents a significant challenge in applications like traffic flow analysis or environmental monitoring. While data is generated on each device, privacy concerns prevent its direct sharing. This poses a dilemma:

- **Centralised learning**: Moving all data to a central server for training is infeasible due to privacy constraints.
- **Local learning**: Training models solely on individual devices leads to limited accuracy and inconsistency across the network.

Gossip learning emerges as a powerful solution for collaborative learning in such scenarios to address these challenges. The idea is for local ML models to be disseminated within the network using random walks in parallel, while applying an online learning algorithm to improve themselves, and getting combined via ensemble learning methods. The algorithm is extremely robust, prediction is possible at any time in a local manner, and it has low communication complexity [12].

Gossip learning uses **peer sampling protocols** so that a node can initiate a random walk over the entire network of nodes. Peer sampling in distributed systems refers to the process where each node maintains a dynamic and representative subset of nodes (peers) in a network. This subset, often referred to as a "sample" or "neighbourhood," provides a snapshot of the network's topology and helps facilitate efficient communication, resource discovery, and various distributed algorithms. The goal is to have an up-to-date view of the network without maintaining a full and exhaustive list of all participating nodes, which can be impractical in large-scale and dynamic distributed environments [13].

Overall, gossip learning provides a promising approach for decentralised learning, enabling collaborative model development while maintaining privacy and offering robustness and scalability for large-scale applications. However, it brings new challenges related to model convergence and accuracy as well as communication costs.

2.3 Robust and energy efficient IoT

Energy efficiency is a major concern in the field of IoT, where limited resources and limited power sources require careful management of energy consumption. By focusing on energy efficiency when developing middleware, you can realise the full potential of IoT technologies while minimising the impact on the environment and operating costs. By optimising energy consumption, middleware can reduce consumption, extend device lifetimes and improve scalability. In addition, energy-efficient nodes play a crucial role in supporting ML workflows in IoT systems that utilise ECC. ML algorithms often require significant computing resources, and energy-efficient nodes ensure that these resources are used wisely. By optimising energy usage during ML model inference and data processing tasks, the middleware enables efficient execution of ML workflows without overloading individual nodes. In this sense, a node can be any connected device that has enough resources to run services via a selected virtualisation engine, e.g. Docker containers or WebAssembly runtimes [14].

WebAssembly (Wasm) is a standardised binary instruction format developed for efficient code execution. It serves as a low-level and portable representation of programmes and enables high-performance

execution of applications. Wasm is intended as a common compilation target for different software so that the code can be executed at near-native speed. In this context, Wasm plays a promising role in promoting energy-efficient IoT. This is achieved by optimising code execution, which reduces the memory and computational power requirements of IoT devices [15]. This optimised execution results in lower energy consumption and ensures more efficient device operation. By promoting local computation, the need for extensive data transfers over networks is reduced, which significantly lowers energy consumption, especially in scenarios where data transfer consumes a lot of energy. Its platform independence ensures the smooth execution of applications on different IoT device architectures and promotes standardised and energy-optimised applications. In addition, Wasm's sandbox execution environment helps to effectively manage resources, control unnecessary access and optimise resource usage, contributing to better overall energy management.

With Wasm, ML models can be compiled and executed directly on IoT devices, enabling efficient and decentralised processing. This approach is particularly useful for scenarios where real-time or edge computing capabilities are critical, helping to reduce the need for constant data transfer to centralised servers. Wasm's portability and ability to run code at near-native speeds make it a suitable choice for deploying ML models on a variety of IoT devices, providing opportunities for on-device inferencing and data processing. However, the specific feasibility and performance will depend on the complexity of the ML model, the resources available on the IoT device and the optimisation techniques applied during compilation.

3 Use cases

The following two use cases are proposed and examined to drive the requirements and design of the data-driven orchestration middleware developed within the AIoTwin project:

- Smart City and traffic management
- Smart Agriculture

These use cases are chosen because project partners have existing experience and results in those domains stemming from previous or current research projects.

3.1 Smart City and traffic management

As global forecasts predict [16], cities are experiencing continuous growth in size and population. This rapid urbanisation presents significant challenges for urban life and puts a strain on resources and services such as healthcare, education, infrastructure and transport. To ensure the sustainability of these services, cities need to use innovative approaches to data management that enable better planning of urban infrastructure and predictive maintenance, for example to plan new roads, bridges or pavements to reduce emissions from cars or to create parks and green roofs to reduce temperatures in summer.

A promising approach to improve sustainability of cities is the concept of smart cities. A smart city is an urban area where IoT technologies and data collection help improve the quality of life as well as the sustainability and efficiency of city operations [17]. A data-driven approach leads to an efficient use of

resources and a reduction in environmental impact with positive effects on sustainability, while optimised city management and decision-making through data-driven insights leads to improved operational efficiency. Ultimately, the quality of life is improved through better services and infrastructure, which in turn improves the well-being of citizens.

At the heart of smart city initiatives lies IoT: It is an enabling technology that allows for the pervasive digitization of infrastructure, bridging the gap between physical and digital world. IoT enables ubiquitous connection of devices to the Internet, allowing them to send information to the upper layers of the ECC, as shown in Figure 1, and potentially to receive directions for performing actions. IoT involves the collection of data from the physical world and performing data analytics to extract information from this vast amount of data to support decision and policy making at different city levels.

A smart city is made up of several components [17]:

- Smart transport – reducing traffic problems, such as congestion, pollution, scheduling, cost reduction for public transport, etc.
- Smart energy – reducing the energy consumption of city infrastructure, e.g., city lights, improved energy consumption and automatization of buildings, etc.
- Smart services – maintaining constant supply of water, waste management, environmental monitoring, etc.
- Smart homes/buildings/offices – installing ambient sensors, motion trackers, power/energy consumption, etc.
- Smart health – decreasing cost of healthcare, ensure healthcare is available to as many people as possible utilizing AI, etc.

The focus of AIoTwin will be on smart transport or traffic management.

Transportation systems are among the most important infrastructures in modern cities, enabling the daily commuting and travelling of millions of people. With rapid urbanisation and population growth, transport systems have become increasingly complex. Modern transport systems include road vehicles, rail transport and various shared transport modes that have emerged in recent years, including online ride-hailing, bike-sharing and e-scooter sharing.

In the development and operation of smart cities and intelligent transportation systems (ITSs), traffic states are detected by sensors (e.g. loop detectors) installed on roads, subway, and bus system transaction records, traffic surveillance videos, and even smartphone GPS (Global Positioning System) data collected in a crowd-sourced fashion, while output typically tends to depend on the use-case. Some of the traffic managements examples are:

- Road traffic flow
- Regional taxi flow
- Regional bike flow
- Station-level subway passenger flow
- Road traffic speed
- Road travel time
- Traffic congestion

- Taxi demand
- Bike demand
- Reporting and monitoring traffic accidents

Entities that are typically involved in a traffic management solution include the following:

- Sensors
  - Various sensors can be deployed for detecting traffic movement and density, such as infrared sensors, acoustic sensors, radars, lidars etc.
- Actuators
  - Traffic lights are the most important actuators that control traffic in a smart city. Also, other actuators can help control traffic, such as digital speed limit signs, variable message signs, roadway lights etc.
- Cameras
  - In a smart city, cameras are mostly placed in crossroads or next to streetlights to monitor traffic conditions and accidents.
- Edge nodes
  - Edge nodes are devices deployed close to the source of the data, whether at near or far edge layer of the ECC, to enable local data processing which reduces the latency, saves bandwidth and enhances privacy. Such nodes can be placed in special cabinets near crossroads to support (near) real-time processing which is required for traffic management use cases.
- Cloud servers
  - Servers hosted within cloud provider infrastructure are required to store large amounts of data and handle resource-intensive data processing and analytics or to run ML models.
- ML models
  - ML models are needed in traffic management solutions to extract traffic patterns, predict traffic congestion or detect accidents. The output of these models can be used to optimize traffic control.

3.1.1 Problem definition: Machine learning on large volumes of traffic data

By leveraging algorithms to analyse data patterns and make predictions based on large volumes of traffic data, ML can help optimise traffic management strategies, reduce congestion on the roads and improve overall transportation efficiency. However, the sheer volume and complexity of available city-wide data pose unique challenges for traditional centralised ML approaches. For this reason, we explore two proposed strategies: federated learning and decentralised learning. Federated learning enables model training across distributed devices by using centralised orchestration for model aggregation, while decentralised learning provides a distributed approach to model training by distributing the learning process across multiple nodes or devices without the need for a central aggregator.

3.1.1.1 Proposed strategy: Federated learning for real-time traffic monitoring

In the context of real-time traffic monitoring, FL can help to improve traffic prediction, congestion detection, and overall traffic management strategies in smart cities. By leveraging data from various sensors, cameras, connected vehicles, and other sources, FL enables the development of intelligent traffic monitoring systems that can adapt to dynamic traffic conditions.

In real-time traffic monitoring, data is generated from a multitude of sources, including traffic cameras, road sensors, connected vehicles, and urban infrastructure. FL allows model training to occur locally on these distributed data sources or in a layer close in the ECC hierarchy, ensuring that sensitive data remains on-site and is not transmitted centrally. This approach brings the following advantages:

- Reduced network traffic: As the data is stored close to the source that generated it, network traffic is significantly reduced to cloud servers, especially when large amounts of data are collected and processed, such as for processing high resolution video streams from road cameras.
- Enhanced privacy: Instead of sharing raw data, only model updates and aggregated insights are exchanged between devices, minimizing privacy risks while still enabling collaborative learning.
- Real-time adaptation: By continuously updating and refining models based on new data observations, FL facilitates dynamic adjustments to traffic management strategies, such as signal timing optimisation, congestion detection, and route planning.

3.1.1.2 Proposed strategy: Decentralised GNN for traffic forecasting

The traffic forecasting problem is more challenging than other time series forecasting because it involves large data volumes with high dimensionality, as well as multiple dynamics including emergency situations, e.g., traffic accidents [18]. Time-series data is a sequence of data points indexed or ordered by time intervals, typically used for analysing trends, patterns, and changes over time. The traffic state in a specific location has both spatial dependency, which may not be affected only by nearby areas, and temporal dependency, which may be seasonal. Traditional linear time series models, e.g., auto-regressive and integrated moving average (ARIMA) models, cannot handle such spatiotemporal forecasting problems[18]. ML and deep learning techniques have been introduced in this area to improve forecasting accuracy, for example, by modelling a whole city as a grid and applying a convolutional neural network (CNN [18]. However, the CNN-based approach is not optimal for traffic foresting problems that have a graph-based form, e.g. road networks.

In recent years, Graph Neural Networks (GNNs) have become the frontier of deep learning research, showing promising performance in various applications [18]. GNNs are well suited to traffic forecasting problems because of their ability to capture spatial dependency, which is represented using non-Euclidean graph structures. For example, a road network is naturally a graph, with road intersections as the nodes and road connections as the edges. With graphs as input, several GNN-based models have demonstrated superior performance to previous approaches on tasks including road traffic flow and speed forecasting problems [18].

Many research papers have focused on using a centralised solution for traffic forecasting. Only a few have proposed a decentralised solution using FL [19]. However, FL requires a central trusted aggregator to combine the contributions of different devices, and thus cannot fully eliminate the issues of scalability,

trust, and privacy. Furthermore, these techniques assume that each device has a sufficiently large view of the graph to perform a complete GNN training step, which is especially problematic for deep GNNs that have a large receptive field.

We propose the usage of a decentralised GNN to solve traffic forecasting problem. Decentralised GNN is best used in scenarios where centralised training is not feasible or desirable, e.g. in case of large data volumes. Decentralised solutions allow for real-time processing of data closer to the traffic sources (e.g., sensors, cameras). This reduces latency and enhances the system's ability to respond rapidly to changing traffic conditions [19].

3.1.2 Problem definition: Maintaining QoS of inference service instances running in smart city environments

From traffic management and public safety to resource allocation and environmental sustainability, inference services provide real-time analysis, enabling cities to respond dynamically to evolving conditions. In the applications of traffic management, inference services process data from sensors, cameras, and other sources to predict traffic patterns, optimize signal timings, and reduce congestions. In public safety, these services analyse surveillance footage, detect anomalies, and enhance emergency response capabilities without compromising individual privacy.

In practice, each service is associated to a set of QoS requirements specifying Service Level Objectives (SLOs) that must be met for all clients using the service. These requirements are based on various parameters, including latency, throughput, and security. Maintaining these requirements above a certain threshold is crucial in smart cities. For example, latency can be of great importance when detecting anomalies such as fires or car crashes, throughput is important when a high-resolution video needs to be processed to detect events on cameras, and privacy requirements are important in all smart city use cases, especially in smart homes.

Once models are trained and inference services are deployed across the ECC, the challenge arises on how to ensure continuous data delivery from IoT devices to running service instances in the dynamic edge-to-cloud environment while adhering to specific QoS requirements and balancing the load on service instances. This is not a trivial task as both IoT devices and nodes running the services can change their states and locations, as well as the high probability of the underlying network between them [1].

3.1.2.1 Proposed strategy: QoS-aware load balancing for smart city services

To ensure QoS for clients using inference services within the ECC, one approach is to deploy a proxy on each node within the cluster. Clients requiring a service only need to connect to the proxy assigned to them, usually the one with the shortest network distance. These proxies act as intermediaries and forward all client requests to the appropriate service instances based on a prediction of whether they can fulfil the QoS requirements. In addition, the proxies continuously monitor the QoS perceived by the clients so that they can dynamically adapt to changes within the ECC environment.

In addition, these proxies can facilitate load balancing between a pool of service instances that can fulfil the QoS requirements. By distributing incoming requests across multiple instances, the risk of overload is reduced and the reliability of inference services is improved.

An example of the use of a QoS proxy could be a real-time collision avoidance system for autonomous vehicles. In these systems, cameras are installed at the roadside that send their video streams to the inference service instances running in the ECC to detect accidents or unpredictable events, e.g. a pedestrian suddenly stepping onto the road behind a parked car. The processing of these data streams must be completed within a short time, usually less than 50 milliseconds as stated in [30], as the information about the event must be sent to the nearby vehicles that might be affected by the event. Therefore, a service instance must be running on a node close to the camera and needs to have a place in the processing queue so that a request can be processed immediately. In this case, a device that collects video from the camera only needs to send the video to the proxy, which ensures that the streams are delivered to the inference service instance that can fulfil the QoS requirement for a service, namely a processing latency of less than 50 milliseconds. The proxy can also perform load balancing on multiple instances that meet the QoS requirements, ensuring that none of these instances are overloaded.

3.2 Smart Agriculture

Smart agriculture [20], often referred to as **precision farming**, is a transformative paradigm for agricultural production that integrates cutting-edge technologies such as IoT, AI and robotics to improve the efficiency, productivity, and sustainability of agricultural practises. At its core, smart agriculture leverages advanced technologies to collect and analyse real-time data from farms [21]. This data-driven approach enables farmers to make informed decisions, optimise resource allocation and accurately monitor crop health. Smart farming encompasses a range of applications, including:

- precision irrigation,
- crop monitoring,
- automated machinery and
- predictive analytics.

By using sensors to collect data on soil moisture, temperature and nutrient levels, farmers can fine-tune their irrigation schedules and fertiliser applications to minimise waste and maximise yields. Drones equipped with cameras and sensors provide a bird's eye view of fields and help with early detection of diseases, pests and other problems [22]. AI algorithms process this data to gain insights that enable farmers to adopt more sustainable and efficient farming methods, leading to higher yields. Overall, smart farming promises to revolutionise traditional farming methods and promote sustainability through resource-efficient food production with less water, fertiliser, herbicides and insecticides.

The integration of advanced technologies into smart agriculture poses a number of challenges that require innovative solutions for sustainable and efficient agricultural practises. One major challenge is the effective management of the extensive data generated by various sensors and devices in agriculture. The sheer volume of data, ranging from soil moisture to crop health indicators, makes it difficult to process, analyse and gain actionable insights from this data. The problem goes beyond the sheer volume of data since it is challenging to transmit data from remote fields where network bandwidth is limited. Low Power

Wide Area Network (LPWAN) protocols such as LoRaWAN and NB-IoT are typically applied for precision agriculture applications.

In precision agriculture, the focus is on real-time data collection which is spatially dense, where connectivity and network reliability becomes important, although not as critical as in smart city use cases. In remote agricultural areas, limited network infrastructure can affect the seamless flow of data between devices, influencing the timeliness and accuracy of decision-making processes. It is worth noting that while not as crucial as in urban settings, connectivity and network reliability continue to play an important role in optimising precision agriculture.

Smart agriculture relies on accurate data to make decisions. Inaccuracies or interruptions in data collection due to unforeseen events can impact the effectiveness of decision-making in agriculture, especially considering that thresholds for taking measures are sometimes below 1 °C (frost protection, evapotranspiration-based irrigation), as noted in [31].

3.2.1 Problem definition: Data transmission optimisation and energy efficiency in agricultural IoT systems

The efficient transmission of data in agricultural IoT systems is a major problem where bandwidth limitations and power consumption are the most important resources. Various solutions have been explored to reduce the amount of transmitted data from the sensors and machines placed in the fields to the processing services, e.g., compression strategies and the use of ML models to predict measurements.

As part of our research objectives, we aim to **improve data transmission processes to increase efficiency** while maintaining the accuracy and integrity of the transmitted data. The biggest challenge lies in managing the large amounts of data generated by various sensors and devices in smart agriculture, potentially overwhelming IoT networks. Traditional real-time transmission of large datasets can lead to latency, increased operational costs and network congestion. To overcome these challenges, we are exploring technologies such as edge computing and AI.

At the same time, energy consumption and efficiency of IoT devices are of the greatest importance. These devices, which include sensors, actuators, smart devices and more, form the backbone of connected systems and enable data collection, processing, and automation. Understanding and optimising the energy consumption of IoT is very important for various reasons. Energy consumption has a direct impact on the **operating costs** and **longevity of the devices**. Optimising energy consumption ensures longer device life and reduces the frequency of replacement and maintenance cycles. It also contributes to cost savings by minimising power requirements, especially in scenarios where devices are operated in remote or off-grid locations and rely on batteries or alternative power sources.

Moreover, energy efficient IoT devices play a critical role in ensuring the reliability and scalability of connected systems. In large-scale deployments where numerous devices communicate and interact with each other, optimising energy consumption is critical to maintaining seamless operations and ensuring consistent performance. By employing energy-efficient components, protocols, and power-saving strategies, these devices can minimize power consumption without compromising their functionality.

3.2.1.1 Proposed strategy: Machine learning at the edge for data filtering

Our proposed approach utilizes ML models deployed on edge devices to perform predictive data filtering in smart agriculture. This is expected to enhance data transmission efficiency by reducing the transmission load on the network, while simultaneously ensuring the accuracy of the transmitted data. Our investigation also targets power consumption of IoT devices, aiming to optimize energy patterns using WebAssembly (Wasm). Wasm's format supports energy-efficient IoT operation, reducing memory and processing power requirements. The decision to focus on smart agriculture is rooted in the sector's growing dependence on data-intensive technologies. Typical devices applied for precision agriculture are resource-constrained, with limited bandwidth and energy source, which creates specific challenges for data transmission and application of ML models in IoT solutions for agriculture. These include the use of remote sensors, resource constraints and the need to make real-time decisions to optimise crop yields.

The proposed approach illustrated in Figure 2 involves deploying ML algorithms on IoT or edge devices to predict and filter redundant data and transmit only relevant information to central processing units located in the cloud. Specifically, the predicted values are calculated both locally and in the cloud. Furthermore, new data readings are transferred to the cloud if a significant deviation between the newly captured data and the predicted value is detected. This approach enables the exclusive transmission of poorly predicted values and thus increases the efficiency of data transmission, especially in the context of precision agriculture. In addition, a balance is achieved between minimising the data transmission load and ensuring the accuracy and reliability of the transmitted data.

Our second goal in this use case is to measure the power consumption of IoT devices and accurately identify the most important power consumers. Through monitoring and analysis, we aim to create a detailed profile of the power consumption patterns of these devices, allowing a better understanding of the power-hungry components and their respective contribution to the overall energy consumption.

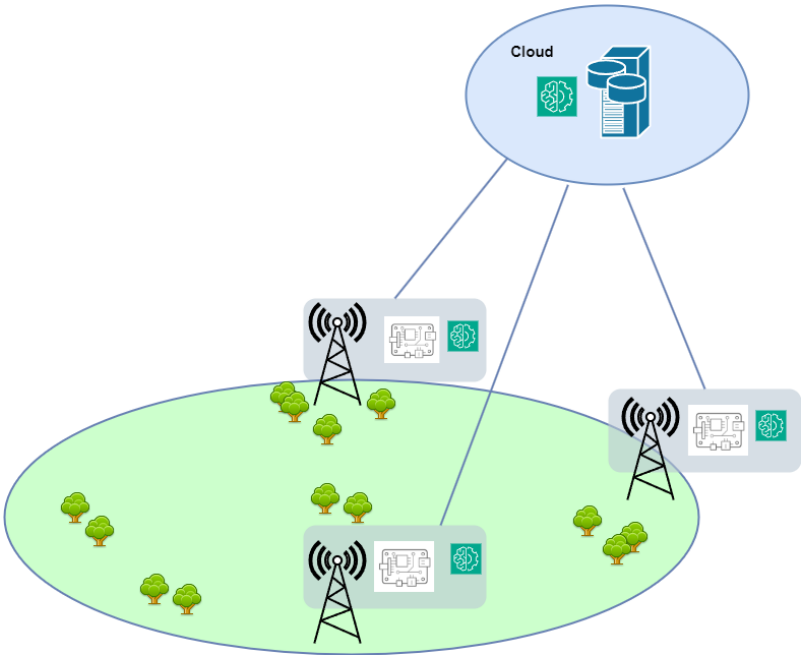

Figure 2. Data filtering on edge devices for precision agriculture

3.3 Available datasets

3.3.1 Traffic management

There are a lot of open data sources for traffic management problems. Data types are categorized in 10 major types [18]

- Transportation network - represents the underlying transportation infrastructure, e.g., road, subway, and bus networks, usually obtained from government transportation departments or extracted from online map services.
- Traffic sensor data - traffic sensors, e.g. loop detectors, are installed on roads to collect traffic information, e.g., traffic volume or speed.
- GPS trajectory data - the trajectory data calculated from GPS coordinate samples can be matched to road networks and further used to derive traffic flow or speed.
- Location-based service data - GPS function is also embedded in smartphones, which can be used to collect various types of location-related data, e.g., check-in data, point-of-interest data, and route navigation application data.
- Trip record data - departure and arrival dates/times, departure and arrival locations, and other trip information.
- Traffic report data - often used for abnormal cases, e.g., anomaly report data and traffic accident report data. Traffic report data are less used in graph-based modelling because of their sparsity in both spatial and temporal dimensions.
- Multimedia data - used as an additional input to deep learning models or for verifying the traffic status indicated by other data sources.
- Simulated traffic data - traffic simulators, such as MATES (The Macro Agent Transport Event-Based Simulator), are used to build virtual training and testing datasets for deep learning models.
- Weather data - Traffic states are highly affected by the meteorological factors including temperature, humidity, precipitation, barometer pressure, and wind strength.
- Calendar data - includes the information on weekends and holidays.

For traffic sensor data type, there are several relevant open datasets available for research use:

- METR-LA - contains traffic speed and volume collected from the highway of the Los Angeles County Road network, with 207 loop detectors. The samples are aggregated in 5-minute intervals. The most frequently referenced time period for this dataset is from March 1st to June 30th, 2012.
- Performance Measurement System (PeMS) data – contains raw detector data from over 18,000 vehicle detector stations on the freeway system spanning all major metropolitan areas of California from 2001 to 2019, collected with various sensors including inductive loops, side-fire radar, and magnetometers. The samples are captured every 30 seconds and aggregated in 5-minute intervals. Each data sample contains a timestamp, station ID, district, freeway ID, direction of travel, total flow, and average speed. Different subsets of PeMS data exist, such as:
    - PeMS-BAY - contains data from 325 sensors in the Bay Area from January 1st to June 30th, 2017.

- PeMSD3 - uses 358 sensors in the North Central Area. The frequently referenced time period for this dataset is September 1st to November 30th, 2018.
- PeMSD4 - uses 307 sensors in the San Francisco Bay Area. The frequently referenced time period for this dataset is January 1st to February 28th, 2018.
- PeMSD7 - uses 883 sensors in the Los Angeles Area. The frequently referenced time period for this dataset is May to June 2012.
- PeMSD8 - uses 170 sensors in the San Bernardino Area. The frequently referenced time period for this dataset is July to August 2016.

- Seattle loop – it was collected by inductive loop detectors deployed on four connected freeways (I-5, I-405, I-90, and SR-520) in the Seattle area, from January 1st to 31st, 2015. It contains the traffic speed data from 323 detectors. The samples are aggregated in 5-minute intervals.

For handling image datasets related to traffic, we can utilize the Vision Knowledge Graph (VisionKG) system developed by TUB [32]. VisionKG is designed to streamline computer vision tasks specifically related to traffic scenarios, such as autonomous driving and pedestrian detection. These tasks rely heavily on diverse and well-annotated datasets to effectively train deep learning models. VisionKG fulfils this need by providing a unified framework that integrates and links various prominent datasets tailored to these applications. These datasets, such as BDD100K, KITTI, MS-COCO, Open Image Dataset and VOC, provide a comprehensive collection of labelled images covering various traffic scenes, objects and signage. VisionKG enables researchers to investigate the relationships and potential biases between these datasets. It gives them easy access to relevant subsets of data for specific tasks such as analysing traffic flow or recognising signs. In this way, VisionKG reduces the time and effort required for data acquisition and encourages the reuse of data by researchers and practitioners in the field.

There is a limited number of datasets that are explicitly designed for federated learning on smart city data. However, any open dataset can be adapted to federated learning scenario by splitting it into smaller datasets based on the data source as long as it has sufficient number of samples per client so that the training can produce a good model. For example, OpenAQ provides open access to air quality data from various locations around the world. It can be used to extract data from different sources and train the air quality models with FL based on data locality.

3.3.2 Smart agriculture

3.3.2.1 Precision Agriculture Monitoring: Crop and field data

Field monitoring for precision agriculture occurs through three primary categories:

1. **Open datasets**: Open datasets such as ERA5-Land and Agri4Cast provide valuable resources for researchers in agriculture and environmental sciences. These datasets are characterised by large spatial coverage and relatively low cost, making them accessible for various applications. However, it is important to note that they come with trade-offs, including lower measurement accuracy and lower temporal resolution.
2. **Proximal Monitoring:** To overcome the limitations of open datasets, the adoption of proximal monitoring techniques has increased significantly, particularly through the use of unmanned

aerial vehicles (drones). Drones provide a more detailed and accurate measurement of environmental variables offering controlled temporal resolution. This capability makes them well suited to tasks that require detailed and timely information, such as crop monitoring and precision agriculture.

3. **Close-Range Monitoring:** Moving even closer to the ground, close-range monitoring involves the deployment of agrometeorological stations and interconnected IoT devices in the field. These stations, equipped with sensor nodes, provide continuous and detailed measurements of key environmental parameters, including soil and air conditions. This approach offers high measurement accuracy but is associated with higher costs and is limited by its coverage area.

The choice of monitoring method used depends on the specific requirements of the application in precision agriculture. Open datasets offer a broad perspective with cost efficiency, while proximal monitoring with drones provides more detailed and accurate information. Close-range monitoring with agrometeorological stations and IoT devices offers the highest precision, but at a greater expense and within a limited spatial area. The combination of these approaches enables a comprehensive and adaptable strategy for monitoring and managing agricultural and environmental conditions.

3.3.2.2 Proposed dataset

Over a three-year period, time series data was collected from several stations across Croatia. This resulted in two primary datasets created by the UNIZG-FER team during the projects Pinova [31] and IoT-Field[1] that provide unique insights into agricultural conditions and plant health.

The first dataset focuses on **agrometeorological parameters** and captures key environmental factors such as temperature, humidity, air pressure, soil temperature and leaf wetness. The second data set contains readings from **multispectral sensors**, a sophisticated technology that measures different wavelengths of light reflected by plants. These readings are then used to calculate important vegetation indices that provide information about plant health and development. The inclusion of multispectral data in the project enhances the ability to assess and monitor vegetation dynamics.

The integration of specialized analytical services increases the utility of the datasets. For example, Growing Degree Days (GDD) calculations contribute to the understanding of the cumulative heat units available for plant growth, aiding crop management decisions. Normalised Difference Vegetation Index (NDVI) analysis provides a quantitative measure of vegetation condition and provides information on plant vitality and stress levels. In addition, yield estimation services improve forecasting capabilities and enable informed decisions on crop productivity.

By combining agrometeorological parameters, multispectral sensor measurements and advanced analytical services, the two datasets provide a solid foundation for smart agricultural practises.

There are two distinct types of stations, each of which collects different parameters. The first type, the called "PIO stations", gathers a wider range of parameters. In addition to the typical agrometeorological measurements such as temperature, humidity and etc., these stations also capture light intensity at

[1] https://iot-polje.fer.hr/iot-polje/en

different frequencies. This measurement is particularly valuable in smart agriculture for calculating the NDVI (Normalised Difference Vegetation Index). The PIO stations are stationed in Slavonia.

The other type of stations, called "FER stations", consists of two stations in Zagreb, at the Faculty of Electrical Engineering and Computing. These stations collect the measurements listed below in Table 1.

Table 1. List of collected parameters

From PIO stations:	From FER stations:
RSSI	Wind Speed
SNR	Battery Level
Air Pressure	Humidity
Air Temperature	Leaf Wetness
Atmospheric Pressure	Solar Radiation
Battery Level	Rainfall
Battery Voltage,	Air Pressure
Compas Heading	Soil Temperature
Wind speed,	Air Temperature
Illumination	Soil Moisture
Irradiation	
Light Intensity on different frequencies	
Lightning avg. distance	
Lightning Strike Count	
Precipitation	
Rainfall	
Relative Humidity	
Solar Radiation	
Wind Direction	

4 System Requirements

4.1 Methodology

The specification of the data-driven orchestration middleware requirements followed a process whose purpose was to derive the key requirements from research problems and specific use cases. In this process, special attention was paid to finding use cases that incorporate different research problems, which will support additional use cases not currently considered within the project.

The iterative process included the following steps:

Step 1: Define requirements for a general architecture.

We are defining initial requirements for the middleware taking into account what is needed for different use case, and it is generic. Such requirements bear the potential of leading to a more efficient architectural design that identifies key functional components across the considered use cases promoting a modular design.

This step is carried out asynchronously with the help of the MS Teams collaboration tool.

Step 2: Introduce use case specific requirements.

The introduction of use case specific requirements starts with analysing research problems and their application to specific use case. Everything that is missing in general requirements is put into a use case specific requirement.

Upon the completion of this stage all partners inspect the derived requirements providing additional input in the form of:

- Additional requirements that are missing from the previous step.
- Assessment on whether a requirement derived by one use case also pertains to other. This also includes comments on the generality of the introduced requirements.
- Any other comment, including comments regarding the precise specification of the intended meaning.
- This step is carried out asynchronously with the help of the MS Teams collaboration tool. At this stage, the Task Leader of T1.4 consolidates all comments and identifies grey areas to be discussed.

Step 3: Finalize requirements.

This last iteration step aims at finalising all requirements ensuring the description is precise, the associated set of use cases has been correctly identified and the importance level within the overall project efforts has been correctly and realistically specified. If some requirements of specific use case are also important for other use cases, then they should be declared as requirements for the general architecture.

4.2 Specified requirements

Table 2 lists the set of requirements specified for the orchestration middleware, each appropriately annotated with its attribute values.

We are using the following acronyms:

- For Type:
  - F – functional,
    - Categories: Interface, Monitoring, Management, …
  - NF – non-functional
    - Categories: Performance, Security, Privacy, …
- For Importance: M – must, S – should

Table 2: Middleware requirements

No.	Type	Category	Importance	Description	Use cases
1	F	Management	M	Middleware should efficiently manage and monitor resources on each node.	1 and 2
2	F	Management	M	Middleware should collect the distribution of data available on node for ML.	1
3	F	Management	M	Middleware should collect the information on the underlying network connecting nodes in the ECC.	1 and 2
4	F	Management	M	Middleware should deploy and manage services across the ECC.	1 and 2
5	F	Management	M	Middleware should be able to run a configuration model to output configuration of a ML pipeline.	1
6	F	Management	M	Middleware should deploy ML components based on a learning configuration.	1 and 2
7	F	Monitoring	M	Middleware should monitor learning performance.	1
8	NF	Performance	M	Middleware should reconfigure the learning pipeline if a better learning performance can be achieved.	1
9	F	Management	M	Middleware should deploy and manage inference components.	1 and 2
10	F	Monitoring	M	Middleware should monitor inference accuracy.	1 and 2
11	F	Monitoring	M	Middleware should monitor inference service performance.	1 and 2
12	NF	Performance	M	Middleware should maintain inference performance and dynamically adapt to changes in the system.	1 and 2

13	NF	Performance	M	Middleware should maintain a desired QoS for clients using the inference services.	1

5 Architecture

We first define a general architecture for orchestration of ML pipelines which includes generic components needed for both training and inference. The general architecture is further refined and accustomed to the requirements of specific use cases. Not all components from the general architecture are needed in specific use cases.

The template for component description is given in Table 3, in compliance with the template recommended in IEEE STANDARD 1016: Software Design Specification [23]. In this document we focus on component descriptions, list of features and related requirements, while detailed component design will be provided in the second and final version of this deliverable.

Table 3. Template for component description

Component	Name of the component
Description	Short description of the component
Provided functionalities	List of functionalities provided by this component
Relation to other components	How will this component interact with other components?
Related use cases	Use cases in which this component is applied
Related requirements	List of requirements that are addressed by this component (Table 2)

5.1 General architecture for orchestration of ML pipelines

WP1 will develop an original data-driven orchestration middleware for ECC to support ML workflows with appropriate data provided by IoT devices, which in turn will be optimised for energy efficiency. The middleware will include mechanisms for service orchestration needed to schedule, deploy and manage management of services in a distributed ECC. Special focus is paid to data routing as one of the offloading criteria, which considers data streams and the need to disseminate them to adequate containers. When orchestrating services in a cloud environment, latency and load balancing are at the centre of most related work, which leaves a lot of room for the development and testing of mechanisms focused on data routing.

The middleware should facilitate energy-efficient operation of low-power end-devices in the IoT-environment. Different mechanisms will be explored which try to find a balance between energy, cost, and performance. Two main strategies can be employed to achieve this goal: adapting monitoring intensity based on energy constraints and desired accuracy levels, and trading latency for energy savings. Additionally, different models for achieving energy efficiency will be analysed (e.g., transmission power modelling, energy harvesting models, etc.).

Mechanisms for distributed and federated learning in the ECC. Since different tasks of the ML pipeline in a distributed learning environment can be executed on different nodes, this needs to be considered when

running the orchestration mechanisms developed in T1.1. In this task, federated learning principles and existing mechanisms (e.g., deep neural networks) will be analysed within WP2 activities, and specific ML workflows will be defined for the use cases specified in T1.4. The defined workflows will be integrated into the orchestration mechanisms designed in T1.1.

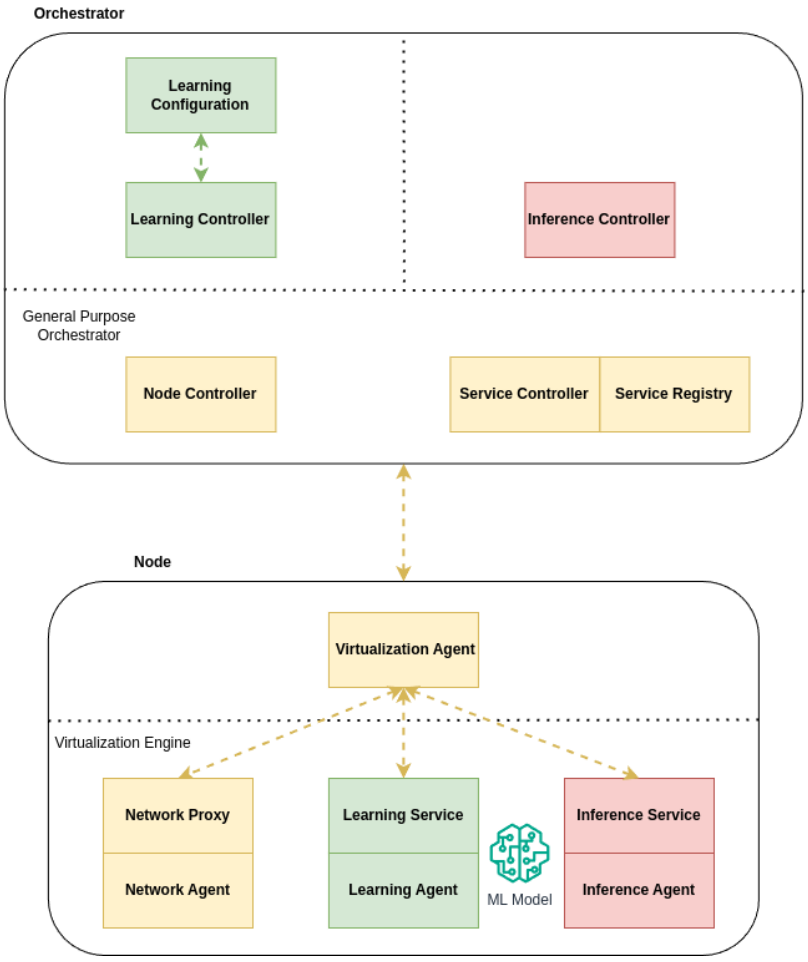

Figure 3. General architecture diagram.

The general architecture of the AIoTwin data-driven orchestration middleware, shown in Figure 3, consists of two main entities: orchestrator and node. Typically, a single orchestrator is used in a centralised setup to orchestrate many nodes offering resources for the deployment and execution of ML pipeline services.

To orchestrate the ML pipeline, the middleware needs to support remote node management and deployment of services across the ECC (requirements 1 and 4). Such essential functionality is integrated within general-purpose orchestrators, such as Kubernetes, which were explained in more detail in Section 2.1. Therefore, this architecture proposes the orchestrator to be designed as an extension of a general-

purpose orchestrator with the components dedicated to managing ML-specific tasks relevant to both learning and inference.

5.1.1 Orchestrator

Orchestrator is the central entity of the proposed architecture and, due to its high-availability requirement, it is deployed in the cloud.

The components belonging to the general-purpose orchestrator, coloured in yellow in Figure 3, are the following: **Node Controller**, **Service Controller** and **Service Registry**. These components enable remote management of nodes and service across ECC. ML-related components, coloured in green in Figure 3, are the following: **Learning Controller** and **Learning Configuration**. These components make sure that a learning pipeline is deployed in the ECC environment and that it adapts dynamically to the changes in the ECC environment. Finally, the only inference-related component, **Inference Controller** coloured in red in Figure 3, manages the inference pipeline and also enables its adaptation to the events in the ECC. Both learning- and inference-related components implement their functionalities in collaboration with the components of the general-purpose orchestrator which provide ECC-related information (up-to-date node and network state) and deploy services of the pipeline in the ECC environment.

Table 4. Node Controller component description

Component	Node Controller (NC)
Description	Node Controller manages and monitors nodes of the system and collects information about node resources.
Provided functionalities	• Node management • Collecting node resource consumption • Collecting network information

Relation to other components	Inbound:
	• Virtualization Agent
	  ○ Sends node state and resource information to NC
	• Network Agent
	  ○ sends network information to NC
	• Service Controller
	  ○ Obtains node information from NC
	• Learning Controller
	  ○ Obtains node and network information from NC
	• Inference Controller

	○ Obtains node and network information from NC Outbound: • Node ○ NC sends message to start the Virtualization Agent
Related use cases	1, 2
Related requirements	1, 2, 3

Table 5. Service Controller component description

Component	Service Controller (SC)
Description	Service Controller deploys and manages services running on the nodes
Provided functionalities	• Service deployment and management • Service monitoring
Relation to other components	Inbound: • Learning Controller ○ Sends requests to SC to deploy learning components • Inference Controller ○ sends requests to SC to deploy Outbound: • Virtualization Agent ○ SC sends requests to deploy services • Service Registry ○ SC obtains service artifacts (more detailed description in Table 6)
Related use cases	1, 2
Related requirements	4

Table 6. Service Registry component description

Component	Service Registry (SR)
Description	Service Registry stores the service artifacts that will be deployed on nodes. Service artifacts are components and resources necessary to package,

	deploy, and run a service within a virtualization environment. The artifacts can contain service code, dependencies, runtime libraries etc.
Provided functionalities	• Storing service artifacts
Relation to other components	Inbound: • Service Controller: ○ Obtains service artifacts from SR
Related use cases	1, 2
Related requirements	4

Table 7. Learning Controller component description

Component	Learning Controller (LC)
Description	Learning Controller deploys components of the learning pipeline, monitors learning and reconfigures the pipeline if needed.
Provided functionalities	• Collecting node resource information and the underlying network characteristics through Node Controller. • Collecting the distribution of data on nodes (in number of samples and class distribution) that can be used for model training from the Learning Agent running on the node. • Obtaining the initial model to be trained from the user. • Running the configuration model to obtain configuration of the learning pipeline. • Deploying learning entities through Service Controller based on the obtained configuration. • System monitoring through Node and Service Controller and rerunning the configuration model upon system changes and, if necessary, deploying new learning components. • Learning performance monitoring obtained by Learning Agent and rerunning the configuration model upon changes in

	performance and, if necessary, deploying new learning components.

Relation to other components	Inbound:
	• Learning Agent
	  ○ sends learning information to LC
	Outbound:
	• Service Controller
	  ○ LC sends requests to deploy learning components
	• Node Controller
	  ○ LC obtains node and network information
	•

Related use cases	1, 2
Related requirements	6, 7, 8

Table 8. Learning Configuration component description

Component	Learning Configuration (LCF)
Description	Learning Configuration component takes inputs (node resources, data distribution, initial model) and outputs the configuration of the learning pipeline, which can contain: roles assigned to nodes (client, local aggregator, global aggregator), local epochs, total number of rounds etc. The architecture is designed in a way that any configuration model can be used depending on the main goal to achieve, such as minimizing training time, resource utilization, communication cost or maximizing performance.
Provided functionalities	• Running the learning configuration model to output configuration of the learning pipeline
Relation to other components	Inbound: • Learning Controller ○ Obtains configuration of the learning pipeline from LCF •
Related use cases	1

Related requirements	5, 8

Table 9. Inference Controller component description

Component	Inference Controller (IC)
Description	Learning Configuration component takes inputs (node resources, data distribution, initial model) and outputs the configuration of the learning pipeline, which can contain: roles assigned to nodes (client, local aggregator, global aggregator), local epochs, total number of rounds etc. The architecture is designed in a way that any configuration model can be used depending on the main goal to achieve, such as minimizing training time, resource utilization, communication cost or maximizing performance.
Provided functionalities	• Collecting node resource information and the underlying network characteristics through Node Controller. • Collecting the distribution of clients that will use the inference service. • Selecting nodes to host inference service based on their resources and the clients that will use the inference service. • Deploying inference service instances through Service Controller. • System monitoring through Node and Service Controller and deploying new instances if needed.

Relation to other components	Inbound:
	• Inference Agent
	  ○ Sends learning information to the IC
	Outbound:
	• Service Controller
	  ○ IC sends requests to deploy inference components
	• Node Controller
	  ○ IC obtains node and network information

Related use cases	1, 2

Related requirements	9, 10, 11, 12

Various system events can trigger a change in the learning pipeline, such as node failures or overload, service failures, or changes in the underlying network like increased latency and limited bandwidth. Also, changes in the learning performance (stragglers, slow accuracy convergence or increasing loss) can result in a new configuration of the learning pipeline.

5.1.2 Node

Node can be any connected device that has enough resources to run various services, including ML pipeline services, through a selected virtualization engine, such as Docker containers or WebAssembly runtimes (WASM). Node components related to the general-purpose orchestrator, which are coloured in yellow in Figure 3, are the following: **Virtualization Agent**, **Network Agent** and **Network Proxy**. Virtualization Agent is in charge of deploying services and reporting service and node states, while Network Agent and Network Proxy are components that collect underlying network information and perform request routing within the ECC. **Learning Service** and **Learning Agent**, coloured in green in Figure 3, are node components related to the learning pipeline where the service is actually implementing the learning task and agent is in charge of monitoring and reporting learning performance to the Learning Controller of the orchestrator. Similarly, **Inference Service** and **Inference Agent**, coloured in red in Figure 3, are components related to inference where Inference Service implements the actual inference task, while Inference Agent monitors and reports inference performance to the Inference Controller.

Table 10. Virtualization Agent component description

Component	Virtualization Agent (VA)
Description	Virtualization agent is run on each node and acts as an intermediate between the orchestrator and the virtualization engine to deploy learning of inference services. An example of a virtualization agent is “kubelet” process of Kubernetes. It also informs the node controller on the node’s resource state.
Provided functionalities	• Running services through virtualization engine • Sending resource state reports • Sending service state reports
Relation to other components	Inbound: • Service Controller ○ Sends requests to VA to deploy services

	Outbound:
	• Node Controller
	  ○ VA sends resource state reports
	• Service Controller
	  ○ VA sends service state reports

Related use cases	1, 2
Related requirements	1, 4

Table 11. Network Agent component description

Component	Network Agent (NA)
Description	Network Agent is deployed on each node to scan the underlying network which includes obtaining network uplink and downlink characteristics and collecting link characteristics on the path to its neighbours. This information is sent to the Node Controller which uses it to create a spanning tree of all nodes and network conditions between them.
Provided functionalities	• Collecting network characteristics on the path to its neighbours • Sending network information reports

Relation to other components	Outbound:
	• Node Controller
	  ○ NA sends network reports
	• Network Agent
	  ○ NA obtains network characteristics with connection to NA's

Related use cases	1
Related requirements	3

Table 12. Network Proxy component description

Component	Network Proxy (NP)
Description	Network Proxy is deployed on each node and it forwards the requests to the inference services based on the specified QoS requirements while performing load balancing.

Provided functionalities	• Forwarding requests to service instances that meet the QoS • Load balancing

Relation to other components	Outbound:
	• Node Controller
	  ○ NP obtains node and network information
	• Service Controller
	  ○ NP obtains available service instances
	• Inference Service
	  ○ NP forwards requests to inference service instances

Related use cases	1
Related requirements	13

Table 13. Learning Service component description

Component	Learning Service (LS)
Description	Learning Service can be any service in the learning pipeline, such as a learning client or a learning model aggregator.
Provided functionalities	• ML • Model aggregation (OPTIONAL)
Relation to other components	Inbound: • Learning Agent ○ Obtains learning performance from LS
Related use cases	1, 2
Related requirements	6, 8

Table 14. Learning Agent component description

Component	Learning Agent (LA)
Description	The responsibilities of the Learning agent may involve tasks such as overseeing the performance of learning processes and informing the Learning

	Controller with information on performance metrics, including but not limited to loss, accuracy, time per round/epoch, etc.
Provided functionalities	• Monitoring learning performance • Sending learning performance reports

Relation to other components	Outbound:
	• Learning Controller
	  ○ LA sends learning performance reports
	• Learning Service
	  ○ LA obtains learning performance

Related use cases	1
Related requirements	7

Table 15. Inference Service component description

Component	Inference Service (IS)
Description	Inference Service is deployed to make predictions based on the pretrained model. Therefore, it is most commonly deployed on the same node that performs learning as it already contains the model, and it can keep the model up to date with the latest version if it is retrained. Inference Service is used by the clients that can come either from inside of the cluster, i.e. other services deployed in the pipeline, or outside of the cluster, i.e. IoT devices that produce the data.
Provided functionalities	• Running predictions (inference)
Relation to other components	Inbound: • Inference Agent ○ Obtains inference performance from IS
Related use cases	1, 2
Related requirements	9, 12

Table 16. Inference Agent component description

Component	Inference Agent (IA)

Description	Inference Agent monitors the inference performance and sends the information to the Inference Controller, such as time per prediction, throughput or prediction accuracy.
Provided functionalities	• Monitoring inference performance • Sending inference performance reports

Relation to other components	Outbound:
	• Inference Controller
	  ○ IA sends inference performance reports
	• Inference Service
	  ○ IA obtains inference performance

Related use cases	1, 2
Related requirements	10

5.2 Adaptive orchestration of FL pipelines

As described in Section 2.2.1 and highlighted in [24], FL has a number of open challenges. One word that might summarize the various challenges in FL is **heterogeneity**. FL Clients participating in training may have different (i) hardware specifications, (ii) network characteristics, or (iii) data distributions. *Hardware heterogeneity* means that training the same model on different hardware will deliver different performances, leading to the occurrence of stragglers. One approach to dealing with stragglers is to offload computations to edge servers [25], but under the important condition that privacy requirements allow data to be offloaded to a nearby server. A highly distributed FL architecture means that participating clients will have *different network characteristics*. Especially in IoT use cases, devices often operate on unstable and bandwidth-limited networks. Therefore, with model sizes of several gigabytes, it is obvious that training performance is highly dependent on communication costs. To address this problem, **hierarchical FL** has been proposed [26]. It places multiple local aggregators at the edge of the network, closer to the FL clients, to perform local aggregation before sending the aggregated models to the global aggregator. With frequent local aggregations, the authors claim that their hierarchical FL approach reduces the overall training time as well as communication and energy costs compared to a traditional cloud-based FL setup. However, the authors did not focus on dealing with stragglers, which was highlighted by [27] where asynchronous global aggregation was proposed. Another important feature to consider when configuring an FL pipeline is *data distribution*. Data is generated by different clients with different quantities and frequencies and is therefore unbalanced and non-Independent and Identically Distributed (non-IID). Although the authors [10] claim that their proposed algorithm FedAvg (Federated Averaging) is robust to non-IID data distribution, several research papers have confirmed that FL will almost inevitably suffer performance degradation due to non-IID data [28]. Therefore, it is important to consider the data distribution on the nodes when selecting nodes to participate in training and, in hierarchical federated learning, to properly balance the clusters, as explored in [29].

The overview of FL challenges brought us to the following conclusions:

- Defining a configuration of a FL pipeline is not a trivial task. It is highly dependent on the model to be trained, the underlying infrastructure, and the data distribution.
- The heterogeneity of the FL environment suggests that changes will occur during the execution of FL. Therefore, the configuration of the FL pipeline will most likely need to be changed during runtime.

An adaptive orchestration mechanism is needed to deploy the entities of the FL pipeline, monitor the execution of the pipeline, and perform reconfiguration as needed. Therefore, in this chapter, we propose an architecture for adaptive orchestration of FL pipelines. Adaptive orchestration is achieved both by predicting future states of the pipeline and by responding to unexpected events.

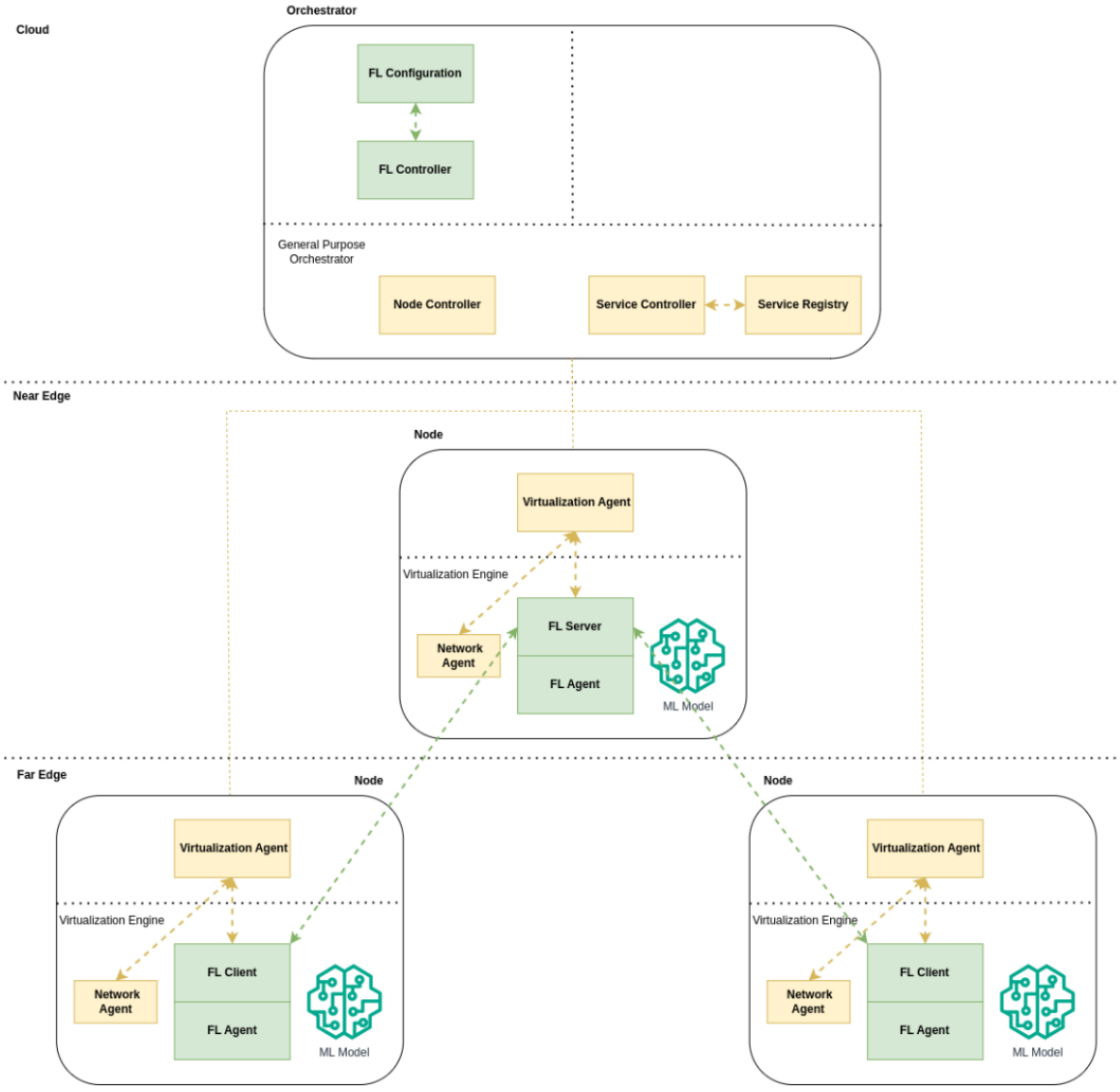

Figure 4. Adaptive orchestration of FL pipelines

The architecture for adaptive orchestration of FL pipelines, shown in Figure 4, is an extension of the general architecture that contains all the general components except the inference-related ones. The specifics of FL are integrated into the **FL Controller**, which, after collecting all the information about the nodes, data distribution and underlying network, executes the **FL Configuration** and deploys the FL components based on its results. **FL Server** is deployed on each node that serves as FL aggregator, be it at local or global level, and it is deployed together with its **FL Agent** that notifies the FL Orchestrator about the FL performance within its cluster, i.e. loss and accuracy in the test set. FL Client runs the client that performs the training and sends model updates to the server. It is also deployed with its FL Agent, which notifies the FL Controller about the training performance, such as training loss and accuracy, as well as the system information collected during the training and the time per training epoch. The remaining components of the general architecture keep the same functionalities as defined in Section 5.1.

5.3 QoS-aware load balancing for inference services in ECC

As described in Section 3.1.2, different service running in the ECC can have different QoS requirements and a problem arises on how to continuously ensure an adequate level of QoS to the clients using the service. Therefore, in this section, to fulfil the requirement #13 from Table 2, we propose **QEdgeProxy**, a distributed QoS-aware load balancer tailored to the ECC. Its primary functions include (i) dynamically maintaining a set of service instances that meet the targeted QoS for a given service, and (ii) forwarding service requests to these instances while performing load balancing. QEdgeProxy serves as a ``QoS agent'' for IoT clients within the ECC, and acts as an external routing component, i.e., an intermediary between IoT clients and IoT services across the computing continuum. QEdgeProxy differs from other edge-aware proxies in that it focuses primarily on adaptively meeting QoS requirements for its clients, rather than solely striving for the best possible QoS. This approach enables the integration of load balancing techniques with request forwarding to mitigate the risk of overloading the processing nodes while adhering to QoS constraints. As QoS largely depends on the processing node where the instance is deployed, and considering that different services can have different QoS requirements, this approach broadens the range of nodes that can be selected for processing a request, potentially leading to enhanced load balancing efficiency.

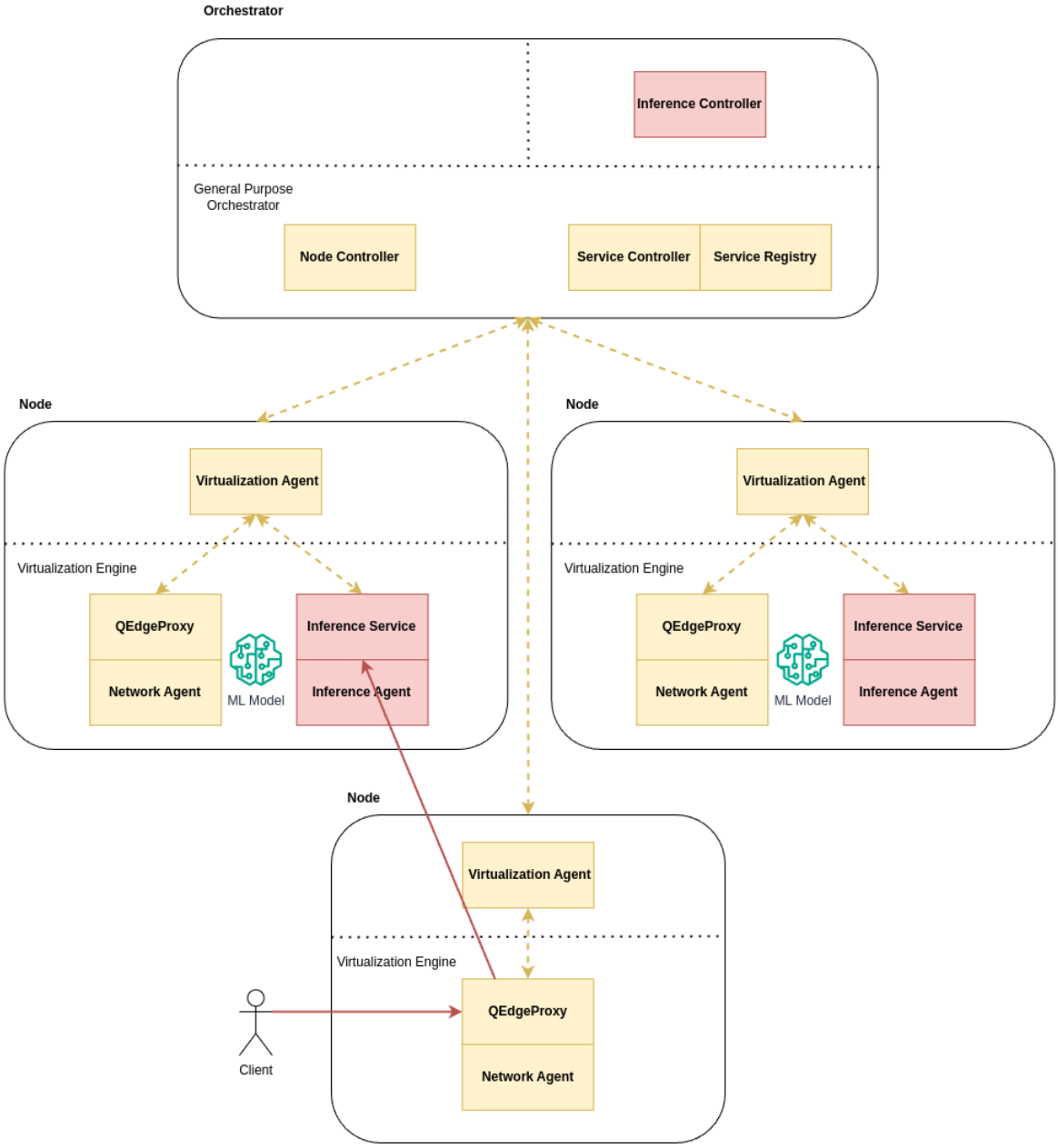

Figure 5 Architecture for QoS-aware load balancing for inference services in ECC

Figure 5 depicts the architecture for QoS-aware load balancing for inference services in ECC as an extension of the general architecture proposed in Section 5.1. The architecture keeps the components related to the general-purpose orchestrator and the inference-related ones. The Inference Controller schedules and deploys inference service instances within the ECC based on the distribution of clients that utilize them, and makes sure that there are sufficient instances to support the demand of the clients. When a client, whether it is an IoT device or another application, wants to start using an inference service,

it connects to the closest QEdgeProxy by network distance. Then it sends its inference request to it, and QEdgeProxy forwards the requests directly to adequate inference service instances.

5.4 Inference for efficient communication and energy-aware edge computing

In this section, we introduce a refined architecture designed to minimize communication overhead and enhance energy efficiency of sensor nodes deployed on resource-constrained devices, while the sensed data needs to be transmitted to the cloud, as defined in Section 3.2.1.1. The proposed architecture consists of two layers, as shown inFigure 6: the Edge Layer, where the devices are deployed, and the Cloud Layer, which hosts the orchestrator and a central node.

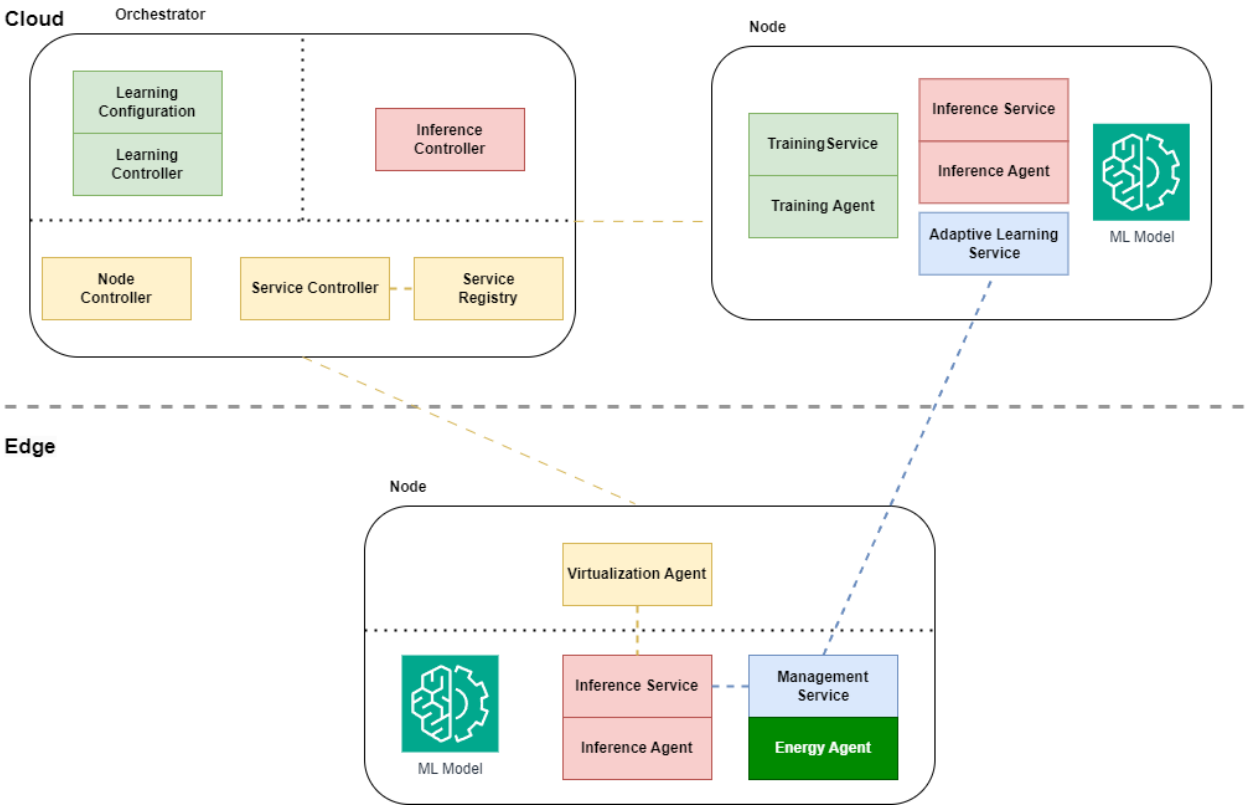

Figure 6. Architecture diagram for efficient communication and energy-aware edge computing

The tasks of the Node Controller, the Service Controller, the Service Registry, the Learning Controller and Virtualization Agent have been previously described in a preceding section and remain consistent within this architecture.

The **Training Service**, which is responsible for model training, is deployed in the cloud together with its dedicated **Training Agent**. This agent monitors the learning performance and forwards performance parameters (loss, accuracy, time per round/epoch) to the **Learning Controller**.

Clients, such as IoT devices producing data, utilize the Inference Service to make predictions from the pretrained model. Each Inference Service comprises an Inference Agent responsible for monitoring performance and communicating information, particularly prediction accuracy, to the Inference Controller. The agent also oversees prediction validation, and if the specified accuracy is not achieved, it notifies the Inference Controller in the Orchestrator. Following this, the Controller notifies the Adaptive Learning Service in response to reported inaccuracies. Upon receiving sufficient data, the Adaptive Learning Service then begins the retraining process.

Table 17. Management Service component description

Component	Management Service
Description	The Management Service is tasked with data collection, efficiently gathering information from the device. The service manages data transmission, ensuring communication during instances of inaccuracies by sending measured values to the central unit. Additionally, it undertakes the role of energy monitoring, keeping a close watch on energy consumption patterns.
Provided functionalities	• Collects data from the device • Manages data transmission during inaccuracies by sending measured values to the central unit. • Monitors energy consumption.
Relation to other components	• Notifies the Cloud-based **Adaptive Learning Service** of inaccuracies if the predefined threshold value is not reached.
Related use cases	2
Related requirements	10

Table 18. Adaptive Learning Service component description

Component	Adaptive Learning Service
Description	Adaptive Learning Service is deployed in the cloud to initiate retraining of a model. It starts training process when a sufficient number of measurements is collected.

Provided functionalities	• Initiates model retraining when a sufficient number of measurements is received.
Relation to other components	• Connects to Training Service to start model training
Related use cases	2
Related requirements	11,12

Table 19. Energy agent description

Component	Energy agent
Description	Energy Agent is deployed at the edge layer and works with the Node Controller to gain real-time insights into the energy consumption patterns of individual devices.
Provided functionalities	• Monitoring the energy consumption of the nodes in the network.
Relation to other components	• Forwards information about energy consumption to the cloud.
Related use cases	2
Related requirements	not related to middleware requirements but it is needed for use case

6 Conclusion

This deliverable focuses on identifying the requirements and defining an initial architecture for a data-driven orchestration middleware for AIoT that proposes the placement of ML services in the edge-to-cloud continuum, taking into account 1) data streams originating from many heterogeneous IoT devices, and 2) the energy consumption of edge orchestration deployments supporting ML workflows. In addition to optimising the placement of containers running AI/ML algorithms considering the available resources, QoS constraints and overall energy consumption, the focus is also on managing ML workflows and data routing in the edge-to-cloud continuum.

The work is initiated by analysing the state of the art of the three specific research domains relevant to the AIoTwin project and the data-driven orchestration middleware: Orchestration in the Edge-to-Cloud Continuum, Federated and Decentralised Learning, and Robust Energy-Efficient IoT. We have selected two areas for the use cases that will be further investigated to be used to test and evaluate the developed

middleware components and libraries. These are traffic management within a smart city context and energy-efficient environmental monitoring and data transmission for smart agriculture. For these two use case areas we detail specific research problems and devise a strategy to approach the problem. The reason for choosing these areas for use cases is that the project partners have experience in developing solutions in the stated areas and are familiar with the available datasets. Use case analysis forms the basis for identifying middleware requirements: 13 requirements have been identified which fall into the category of functional (management and monitoring) and non-functional (performance) requirements. Finally, we present the general architecture four our data-driven orchestration middleware for AIoT. The architecture includes generic components needed for both training and inference in the edge-to-cloud continuum; the components are deployed at both orchestrator and edge nodes. This generic architecture consists of six components for general-purpose orchestration extended by four components specifically tailored for the learning phase and three for inference. The general architecture is further refined and adapted to the requirements of specific use cases where the middleware is envisioned to be used for hierarchical FL and efficient real-time inference, either to guarantee specific QoS to clients in the context of traffic management or to minimise energy-consumption for the smart agriculture use case.

7 Acronyms

AI	Artificial Intelligence
AIoT	Artificial Intelligence of Things
D&C	Dissemination and Communication
ECC	Edge-to-Cloud Continuum
FL	Federated Learning
GNN	Graph Neural Network
GPS	Global Positioning System
IoT	Internet of Things
ITS	Intelligent Transportation Systems
LPWAN	Low Power Wide Area Network
ML	Machine Learning
P2P	Peer-to-peer
QoS	Quality of Service
RSSI	Received Signal Strength Indicator
SNR	Signal-to-noise ratio
Wasm	WebAssembly

8 List of Figures

9 List of Tables

10 References

[1] J. Zhang and D. Tao, "Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things," in IEEE Internet of Things Journal, vol. 8, no. 10, pp. 7789-7817, 15 May, 2021, doi: 10.1109/JIOT.2020.3039359.

[2] S. Duan et al., "Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey," in IEEE Communications Surveys & Tutorials, vol. 25, no. 1, pp. 591-624, Firstquarter 2023, doi: 10.1109/COMST.2022.3218527.

[3] I. Čilić and I. Podnar Žarko, "Adaptive Data-Driven Routing for Edge-to-Cloud Continuum: A Content-Based Publish/Subscribe Approach," in Proceedings of the Internet of Things; Springer International Publishing, 2022

[4] OpenFog Consortium, OpenFog Reference Architecture for Fog Computing, 2017

[5] L. Vaquero, F. Cuadrado, Y. Elkhatib, J. Bernal-Bernabe, S. Srirama and M. Zhani, " Research challenges in nextgen service orchestration," Future Generation Computer Systems, 2019

[6] I. Čilić, P. Krivić, I. Podnar Žarko and M. Kušek, "Performance Evaluation of Container Orchestration Tools in Edge Computing Environments," Sensors, 2023

[7] O. Oleghe, "Container Placement and Migration in Edge Computing: Concept and Scheduling Models," IEEE Access , 2021

[8] R. Vaño, I. Lacalle, P. Sowinski, R. S-Julián and C. Palau, "Cloud-Native Workload Orchestration at the Edge: A Deployment Review and Future Directions," Sensors, 2023

[9] S. Hoque, M. De Brito, A. Willner, O. Keil and T. Magedanz, "Towards Container Orchestration in Fog Computing Infrastructures," in Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC) , 2017

[10]H. B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016
[11]Y. Li, X. Wang, R. Zeng, P. K. Donta, I. Murturi, M. Huang and S. Dustdar, "Federated domain generalization: A survey," 2023
[12]R. Ormándi, I. Hegedűs and M. Jelasity, "Gossip Learning with Linear Models on Fully Distributed Data," Concurrency and Computation Practice and Experience, 2013
[13]M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec and a. M. v. Steen, "Gossip-based peer sampling," ACM Trans. Comput. Syst., 2007
[14]V. Kjorveziroski and S. Filiposka, "WebAssembly Orchestration in the Context of Serverless Computing," Journal of Network and Systems Management. 31, 62 (2023). https://doi.org/10.1007/s10922-023-09753-0
[15]P. P. Ray, "An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions," Future Internet, vol. 15, 2023
[16]J. Gracias, G. Parnell, E. Specking, E. Pohl and R. Buchanan, "Smart Cities—A Structured Literature Review," Smart Cities, pp. 1719-1743, 2023
[17]A. Syed, D. Sierra-Sosa, A. Kumar and A. Elmaghraby, "IoT in Smart Cities: A Survey of Technologies, Practices and Challenges," Smart Cities, no. 4, pp. 429-475, 2021
[18]W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," Expert Systems with Applications, vol. 207, 2022
[19]L. Giaretta and S. Girdzijauskas, "Fully-Decentralized Training of GNNs using Layer-wise Self-Supervision". 2023, Zenodo. https://doi.org/10.5281/ZENODO.8088059
[20]Z. Zhai, J. F. Martínez, V. Beltran and N. L. Martínez, "Decision support systems for agriculture 4.0: Survey and challenges," Computers and Electronics in Agriculture, vol. 170, p. 105256, 2020
[21]R. Singh, R. Berkvens and M. Weyn, "AgriFusion: An Architecture For IoT And Emerging Technologies Based On A Precision Agriculture Survey," IEEE Access, vol. PP, pp. 1-1, 2021
[22]R. P. Sishodia, R. L. Ray and S. K. Singh, "Applications of Remote Sensing in Precision Agriculture: A Review," Remote Sensing, vol. 12, 2020
[23]IEEE Standards Association, IEEE Standard for Information Technology – Systems Design – Software Design Descriptions, 2009
[24]P. Kairouz, Advances and Open Problems in Federated Learning, 2021
[25]Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei and F. R. Yu, "Computation offloading for edge-assisted federated learning," IEEE Transactions on Vehicular Technology, 2021
[26]L. Liu, J. Zhang, S. Song and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in IEEE International Conference on Communications (ICC), 2020
[27]Z. Wang, J. L. H. Xu, H. Huang, C. Qiao and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021
[28]H. Zhu, J. Xu, S. Liu and Y. Jin, "Federated learning on non-iid data: A survey," Neurocomputing, 2021
[29]Y. Deng, F. Lyu, J. Ren, Y. Zhang, Y. Zhou, Y. Zhang and Y. Yang, "Share: Shaping data distribution at edge for communication-efficient hierarchical federated learning," in IEEE 41st International Conference on Distributed Computing Systems (ICDCS), 2021

[30]P. Schulz et al., "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," in IEEE Communications Magazine, 2017

[31]D. Kreković D, V. Galić, K. Tržec, I. Podnar Žarko and M. Kušek "Comparing Remote and Proximal Sensing of Agrometeorological Parameters across Different Agricultural Regions in Croatia: A Case Study Using ERA5-Land, Agri4Cast, and In Situ Stations during the Period 2019–2021, " *Remote Sensing*. 2024; 16(4):641. https://doi.org/10.3390/rs16040641

[32]Le-Tuan, A., Tran, T.K., Nguyen, D.M., Yuan, J., Hauswirth, M., Le-Phuoc, D.: VisionKG: Towards a unified vision knowledge graph. In: ISWC (Posters/Demos/Industry) (2021)

[33]Yuan, J., Le-Tuan, A., Nguyen-Duc, M., Tran, T.-K., Hauswirth, M., & Le-Phuoc, D. (2023). VisionKG: Unleashing the Power of Visual Datasets via Knowledge Graph. arXiv [Cs.CV]. http://arxiv.org/abs/2309.13610

