ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 38

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

A FEDERATED DIGITAL TWIN FRAMEWORK LEVERAGING AI AND MLOPS FOR THE IOT CLOUD CONTINUUM

Presentata da: Muhammad Azaz Farooq

Coordinatore Dottorato

Paola Salomoni

Supervisore

Paolo Bellavista

Co-supervisore

Giuseppe Di Modica

Esame finale anno 2026

Abstract

The increasing complexity of industrial systems and the substantial amount of data generated by sensors and interconnected technologies make manual analysis difficult. Industry 4.0, which incorporates Cloud Computing (CC), Artificial Intelligence (AI), the Internet of Things (IoT), and cyber-physical systems (CPS) into manufacturing, enables Machine Learning (ML) algorithms to automate Anomaly Detection (AD) in industrial processes. ML models can predict equipment malfunctions, detect abnormalities, optimise system configurations, and assess hypothetical scenarios for strategic decision-making. Collaborative computing in IoT-Edge-cloud (IEC) represents a promising solution for the Industrial Internet of Things (IIoT), facilitating the design and implementation of Digital Twins (DTs). DTs are utilised throughout multiple sectors, encompassing product design, predictive maintenance, performance monitoring, supply chain management, and business optimisation. Implementing ML for DT involves issues such as data integration, purification, preprocessing, adaptability, interoperability, and fulfilling diverse integration and development requirements. In this context, an AI-based automated DT framework must be identified and explored to make industrial DT systems more adaptable and interoperable to fulfil the growing AI and MLOPs-based requirements. Similarly, Industry 4.0 also requires safe and efficient data sharing, especially in distributed environments where privacy constraints prevent the sharing of raw data. Federated Learning (FL) offers a solution by enabling collaborative training of models without transferring raw data. Edge devices train local models and share only model updates with the central server, preserving privacy. However, applying FL to real-world industrial scenarios poses challenges due to non-IID, high-dimensional, and heterogeneous data. Industrial systems are typically structured in hierarchical layers, leading to streaming input and missing values that degrade model convergence and accuracy. Current frameworks often fail to adapt to dynamic industrial workloads, resulting in poor generalisation and scalability.

To address these challenges, this dissertation aims to develop a privacy-preserving, automated, interoperable, and adaptable federated DT framework leveraging AI and MLOps in IEC settings. The proposed framework, C2E-AIOps, has been implemented for a specific industrial use case, Marposs S.p.A., verifying its credibility and proof-of-concept. Before starting the framework implementation, several state-of-the-art unsupervised ML algorithms were applied to the industrial dataset. They were deployed and automated in our industrial use case using C2E-AIOps. Data management and ML model training occur at the cloud layer, whilst models are deployed and maintained at the edge. To improve interoperability and portability, trained ML models are transformed into lightweight and portable formats via the Open Neural Network Exchange (ONNX) standard. Continuous Integration and Continuous Deployment (CI/CD) are executed on an industrial edge device with GitHub Actions. RESTful APIs utilize standard HTTP techniques to administer the industrial IoT Central edge manifest. Azure Function has been used to automate the deployment of C2E-AIOps in our industrial use case. Experimental evaluation illustrates the efficacy of C2E-AIOps prototype in a particular industrial application.

Similarly, to address data privacy issues in industrial systems, we proposed FedAdapt-CAD: a novel FL framework designed for AD in industrial control systems

at the edge layer. It incorporates Client-Aware Aggregation (CAA) and Dynamic Model Adaptation (DMA) to dynamically weigh client contributions based on data quality and local model performance. This approach was evaluated on three benchmark datasets, WADI, CMAPSS, and BATADAL, covering diverse industrial and CPS. The results show FedAdapt-CAD achieves superior global accuracy while maintaining fairness across heterogeneous clients. The inclusion of CAA and DMA improves detection rates for rare and complex attack scenarios by up to 15 percent. Thus, proposed C2E-AIOps framework opens new dimensions for automated, scalable, deployment-ready, and privacy preserved industrial solutions. It predict anomalies in advance, managing ML models deployment, and ensuring the privacy of industrial datasets in IEC continuum.

Finanziato dall'Unione Europea - Next Generation EU, Missione 4, Componente 2,
Investimento 3.3 (DM 117/2023)
CUP: J33C22001400009

Acknowledgements

I extend my gratitude to my main supervisor, Prof. Paolo Bellavista, and co-supervisors, Prof. Giuseppe di Modica and Prof. Armir Bujari, for the opportunity to conduct this research and for their invaluable guidance, support, and expertise throughout my PhD journey. Specifically, Prof. Bellavista's open-door attitude facilitated numerous stimulating talks that significantly shaped the trajectory and contributions of this dissertation. He assisted me in every difficult situation and guided well, so I was able to overcome stress, research, and administrative challenges, and focus on my research objectives and complete them on time.

I also extend my gratitude to our industry partner, Marposs S.p.A., especially the R&D manager Andrea Bittasi and my Senior Team Lead Software Architect Alberto Sitta for their valuable support and guidance; without their support, it would not have been possible to complete my PhD and explore new dimensions of research.

I also express my gratitude to Prof. Jamal Abdul Nasir, who is my supervisor during my research period abroad at the University of Galway, Ireland. His unconditional support and open-door policy help me a lot to discuss the research problems and find novel solutions that make my PhD journey more thrilling and exciting.

On a personal note, I wish to express my gratitude to my family, especially my wife, my daughter, my parents, and my relatives, for their continued love and support throughout my PhD journey. I have come to believe that overcoming life's significant problems largely depends on one's capacity to persist, even when the ultimate goal remains unclear.

Contents

List of Figures

List of Tables

Acronyms

AD Anomaly Detection.
AI Artificial Intelligence.
ARIMA Autoregressive Integrated Moving Average.

CAA Client Aware Aggregation.
CI/CD Continuous Integration and Continuous Deployment.
CPS Cyber Physical Systems.

DBSCAN Density-Based Spatial Clustering.
DevOps Development Operations.
DMA Dynamic Model Adaptation.
DS Data Science.
DT Digital Twin.
DTs Digital Twins.

ELMs Extreme Learning Machines.

FL Federated Learning.

IAC Infrastructure as Code.
IEC IoT-Edge-Cloud.
IF Isolation Forest.
IIoT Industrial Internet of Things.
IoT Internet of Things.

JSON JavaScript Object Notation.

ML Machine Learning.
MLOps Machine Learning Operations.
MQTT Message Queuing Telemetry Transport.

NN Neural Network.
non-IID non-Independent and Identically Distributed.

SARIMA Seasonal ARIMA.
SME Small and Medium Enterprise.
SMEs Small and Medium Enterprises.
SPC Statistical Process Control.
SVM Support Vector Machine.

Chapter 1

Introduction

In recent years, many jobs that were previously performed manually have been streamlined with intelligent technologies. In the context of smart manufacturing (Industry 4.0), these technologies facilitate monitoring of production lines and the identification of potential issues before they escalate, resulting in less downtime and increased efficiency [1, 2]. Smart industries are crucial to everyday existence. Disruptions can adversely affect production, resulting in economic losses. Typically, these disruptions arise from diverse forms of unanticipated and unexpected behaviours that might appear in these situations, referred to as anomalies [3]. Anomalies in industrial machinery may arise from equipment faults, environmental factors, and variations in operating conditions. Consequently, early detection of these anomalies is crucial to prevent equipment failure, decrease downtime, and reduce repair expenses [3]. Large-scale data collection from industrial machinery has been made possible by IoT ecosystems, offering a wealth of data for AD [4]. Sensors and monitoring systems that gather information about temperature, pressure, vibration, and power usage comprise these ecosystems. With the use of this data, engineers and experts can monitor the condition of industrial machinery in real-time and act quickly to identify and rectify any issues before they significantly impair productivity. However, it is challenging to manually identify anomalies due to the volume and complexity of data generated by IoT ecosystems, which highlights the need for preventive and corrective maintenance. By examining the data produced by these environments, ML algorithms can assist in automating the AD process [5]. Even though there are several ML algorithms for AD, each has its advantages and disadvantages, and can be applied based on the specific particulars of the industrial setting and the type of data.

The IoT is primarily perceived as a network facilitating data interchange among devices, predominantly influenced by consumer desires, emphasising machine-to-user interactions and client-server relationships [6]. Conversely, IIoT, an essential element of Industry 4.0, focuses on linking industrial assets, including machinery and control systems, to business processes and information systems. The introduction of IoT devices boosts productivity by facilitating connections and data exchange inside production processes [7]. Furthermore, IIoT offers cost-effective updates of current industrial infrastructure by adding sensors to older equipment [8]. When it comes to data volume, IIoT usually involves substantial data transfers, permitting ML-based analysis and maintenance advancements. The authors in [9] illustrate this methodology by employing a variety of integrated IoT sensors that transmit

substantial data via the Message Queuing Telemetry Transport (MQTT) protocol for ML-based AD. Nonetheless, it is essential to acknowledge that IIoT applications often face limitations in power and computational resources [10].

1.1 Research Context and Motivation

ML is a technique that enables computers to perform complex tasks such as prediction, classification, regression, and segmentation. The success of ML is dependent on the availability of high-quality data and large datasets for algorithm training, both of which have become increasingly accessible in the industrial environment since the introduction of IIoT [11]. ML algorithms can be categorised as supervised, unsupervised, semi-supervised, or reinforcement [12]. Our work examines the implementation of unsupervised ML methodologies for AD. Unsupervised algorithms offer a significant advantage by eliminating the need for labelled data during training. This is especially beneficial for identifying flaws in machinery, as fault data is often rare and limited. Figure 1.1 [13] illustrates an example of how IIoT and ML can be utilized in an industrial setting to identify issues and plan maintenance proactively. IIoT devices can be thought of as different sensors that can connect locally. Some of the data generated can be handled on devices near the IIoT sensors. This is called edge processing. This can lower the cost of finding anomalies, speed up the process, and reduce the volume of data transferred to the cloud. However, in our study, AD algorithms require substantial computing resources; therefore, they need to be further optimized to operate effectively in edge environments. The data collected is usually stored in a cloud data center (Azure, Google, or AWS) and used to train ML algorithms. Many ML algorithms require extensive training on large datasets. After training is complete, the AD models detect anomalies. Finally, the inference of the trained AD model is performed at the edge or cloud layer.

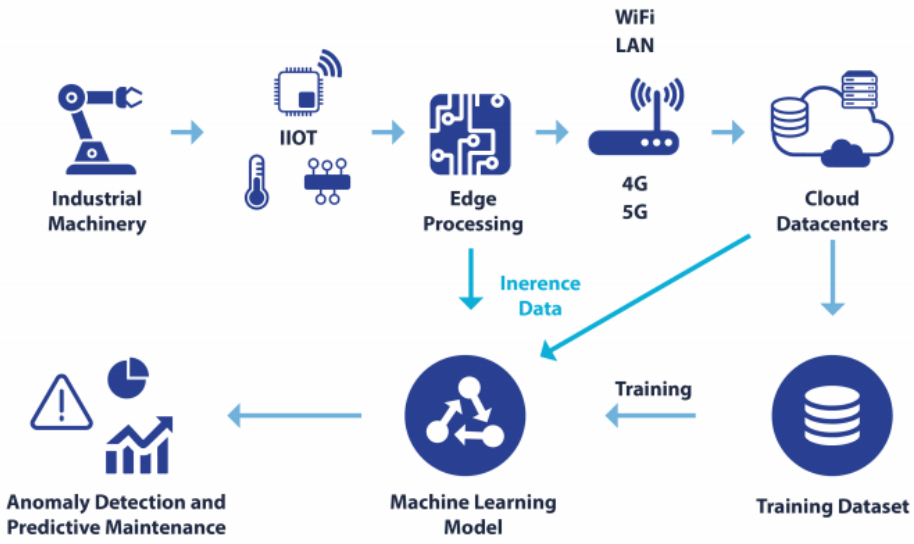

Figure 1.1: An overview of the application of IIoT and ML for AD in industry 4.0

The current manufacturing industry is characterized by highly competitive [14] and increasingly dynamic markets [15]. Innovative consumption models necessitate

that consumers demand highly tailored products [16] and require expedited market entry [17], compelling the manufacturing sector to swiftly adjust to fluctuating demand and to rapidly produce diverse product choices. Changing consumer behavior, along with recent technological advancements, has motivated organizations to pursue digitalization by integrating new technologies to enhance productivity and efficiency [14, 18]. Companies realize the productivity potential that emerging manufacturing technologies, system connectivity, and digitalization offer, facilitating more effective and sustainable resource utilization and the development of new business models [16]. The virtual world is becoming increasingly significant in manufacturing at the same time. The interconnections between the physical and virtual realms, along with their fusion and seamless integration, are recognized as a crucial trend that facilitates technological advancements in product development and manufacturing [19]. Technologies deployed to address the integration of the physical and virtual realms in Smart Manufacturing include Big Data, Artificial Intelligence (AI), IoT, IIoT, edge computing, and cloud computing. These technologies have emerged in the manufacturing sector over the past few years. Simultaneously, global initiatives and plans for manufacturing development have been launched as part of a subsequent phase in transforming the manufacturing sector, highlighted as Industry 4.0 [20].

Smart manufacturing and Industry 4.0 have become revolutionary strategies that are transforming conventional production methods. The fourth industrial revolution, also known as Industry 4.0, is characterised by the integration of cloud computing, AI, IoT, and CPS into manufacturing processes. It encourages the development of autonomous networks with real-time communication, data analysis, and decision-making capabilities. As a key component of Industry 4.0, smart manufacturing aims to optimise production processes using cutting-edge technologies and data-driven tactics. To facilitate proactive decision-making and enhance overall production performance, it leverages automation, AI, Big Data analytics, and ML. This integration of technologies is necessary to achieve smart manufacturing. Additionally, connectivity is crucial for enabling seamless communication between systems and devices. By allowing the connection of sensors, devices, and components, the IoT facilitates the sharing of real-time information and insights. Another essential element of smart manufacturing is the collection and analysis of data. The widespread use of sensors and networked technologies generates massive volumes of data. Predictive analytics is enabled by advanced analytics tools that help identify patterns and extract valuable information. This allows producers to find bottlenecks or inefficiencies, optimise operations, and make well-informed decisions.

The growing complexity and spread of industrial systems, along with the vast volume, diversity, and velocity of data produced by sensor readings, operating logs, environmental parameters, and user interactions, surpass the capacity for manual analysis. Conventional threshold-based monitoring methods frequently lack adaptability to changing conditions and the capacity to detect nuanced patterns indicative of upcoming events. In this context, ML emerges as a crucial facilitator in converting raw data into meaningful insights. [21]. ML models can forecast equipment failures, identify anomalies indicating inefficiencies or faults, optimize system settings for reduced energy consumption, and evaluate hypothetical scenarios to inform strategic decision-making. [18, 22, 23]. When selecting and implementing an effective AD technique, several key aspects must be considered, including the characteristics of the generated sensory data stream, the type of anomaly, and the

availability of training data. Figure 1.2 [24] illustrates a methodological flowchart of the AD system.

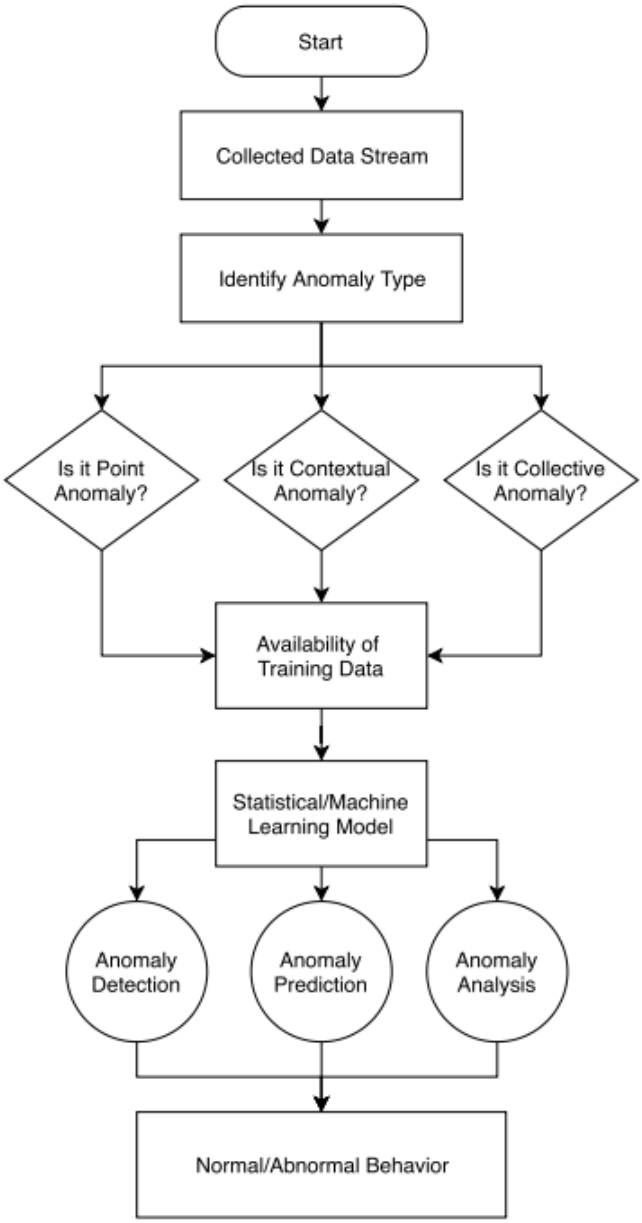

Figure 1.2: AD systems flowchart

In an AD system, shown in Figure 1.2, the initial stage involves understanding the characteristics of the acquired data stream. It may be binary, continuous, or discrete, and may be accompanied by a relational structure. This relational framework encompasses time series, spatial, and graph data. This relationship helps in selecting the appropriate technique for AD analysis or prediction. The subsequent stage involves categorising the anomaly into a predetermined classification: point anomaly, contextual anomaly, or collective anomaly [25]. The third step is to ascertain the availability of training data for constructing an AD system. Based on the availability of data and its annotation, it may be classified as supervised, unsupervised, or semi-supervised. This information helps engineers select a suitable AD technique. Supervised training utilises labelled data, representing the most basic form of learning

to identify anomalous system behaviour. In unsupervised learning, data is available without definitive outputs (i.e., class labels). In semi-supervised learning, there is a limited number of instances with class labels, while the remainder of the data is unlabelled. In our industrial use case, we have an unlabelled dataset and used unsupervised ML algorithms to detect anomalies. Clustering techniques (K-Means Clustering), Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Isolation Forest (IF) have been implemented on specific industrial DT machines. Their implementation details and results will be discussed in the next chapters.

IEC collaborative computing is considered one of the most promising technologies for the IIoT, providing efficient methods for managing computationally intensive and time-sensitive tasks [26]. IoT layer devices can transmit data to systems at other layers, including the cloud, via the Internet. The cloud computing model organizes data centrally within cloud data centers. This method of processing, evaluating, and storing data fails to satisfy the varied requirements of IoT applications. The remote sites of cloud data centers require data transfer with IoT devices, potentially leading to network bottlenecks, severe congestion, and increased latency. Consequently, novel computing paradigms, such as edge computing, have arisen as an intermediary layer between IoT and cloud infrastructures to address these challenges and deliver services closer to IoT devices. The integration of IoT, edge, and cloud layers enables effective, real-time data handling across many applications and use cases. This phenomenon is referred to as the Computing Continuum, in which computing is distributed across the many layers of the continuum. Consequently, high-performance, scalable infrastructure and exceptional dependability in clouds, along with low-latency, privacy-preserving computation at the edge, can be achieved. Such a computing system continuum is illustrated in Figure 1.3 [27].

Leveraging the benefits of IEC while avoiding its constraints, innovative DTs can be designed and implemented. A DT is a virtual replica of a real-world asset, such as a system, machine, or product. It creates a virtual counterpart that replicates the real object by fusing real-time data from sensors, IoT devices, and other sources with sophisticated analytics and simulation tools. Before making adjustments in the real world, manufacturers can use this virtual representation to test scenarios, predict and prevent problems, and monitor, evaluate, and improve performance. The DT provides useful information in a consistent format, enabling smart manufacturing. Michael Grieves first proposed the idea of DT in 2003, marking the beginning of its development. One of the first organisations to use DT was NASA, which used it to track the flight characteristics of a rocket-shaped 'flying twin' [28]. Afterwards, definitions of DT emerged in contexts beyond aerospace, which include a broader range of products and production resources [29]. Since then, DTs have been utilised in various industries, including nuclear reactors, greenhouses, mine ventilation, manufacturing, and virtual PLCs [19, 30–32]. However, in recent years, DTs have become particularly prevalent in the manufacturing sector, where they are crucial to advancing smart manufacturing projects [33]. The primary driving force for employing a DT is to achieve enhanced efficiency, economic benefits [14], improved operational safety, superior product quality, and increased productivity [34]. The variety of application domains has led to a scattered definition of DT; however, the term's ambiguity is not the sole barrier to implementing DTs. The management of large data volumes, forecasting complex systems, and limited synchronization capabilities between the physical and digital realms present further implementation challenges

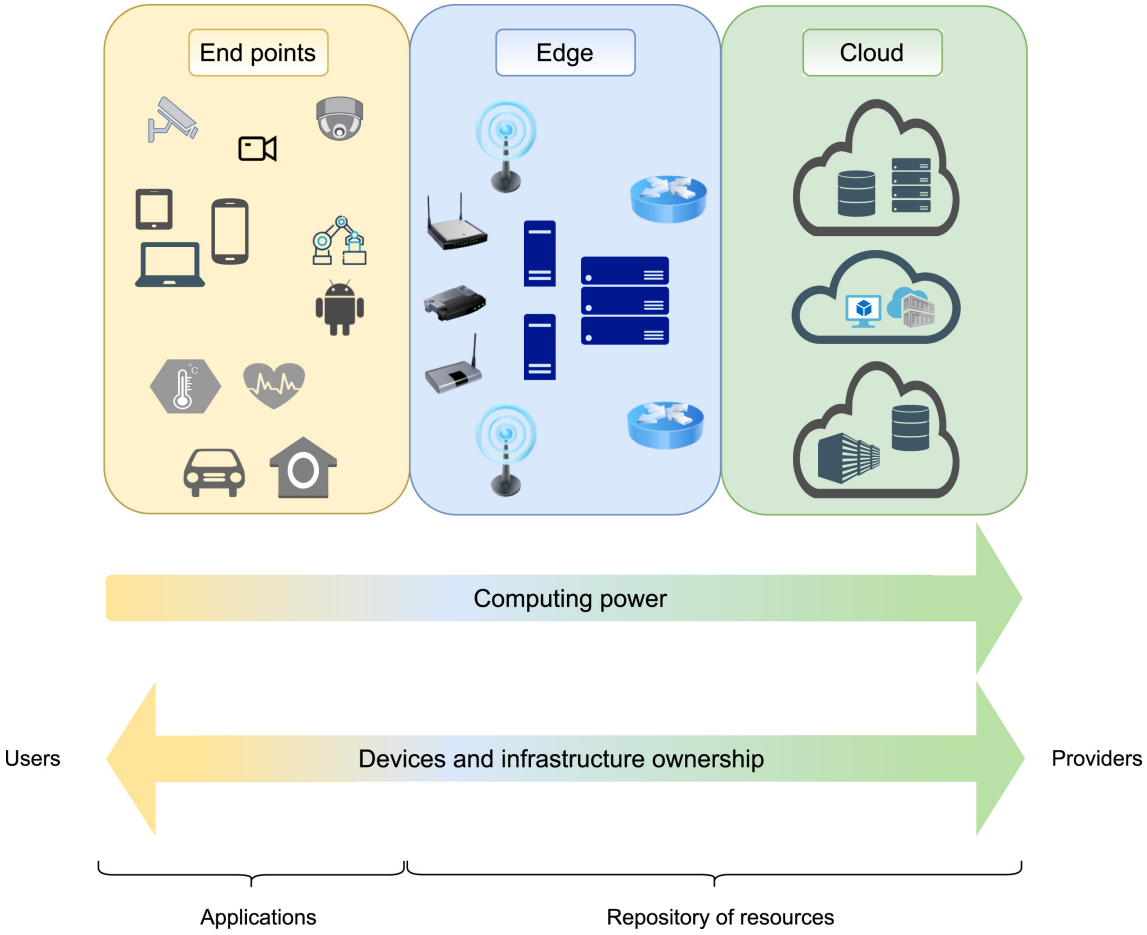

Figure 1.3: The Iot-Edge-Cloud computing continuum

[17]. The International Organization for Standardization (ISO) is now undertaking standardization work on DT in manufacturing [34], and growing awareness of the topic has facilitated the definition and categorization of DT.

With the broad adoption of IoT and Big Data, DT technology is experiencing significant popularity. A recent analysis by Markets and Markets [35] indicates that the DTs market, estimated at USD 6.9 billion in 2022, is anticipated to reach USD 73.5 billion by 2027, reflecting a remarkable compound annual growth rate of 60.6 percent over 5 years. This analysis identifies critical industries likely to make significant investments in this technology, including Automotive and Transportation, Energy and Utilities, Infrastructure, Aerospace, Healthcare, Petroleum, and Natural Gas. These industries are utilising DTs in many applications, including product design and development, predictive maintenance, performance monitoring, supply chain management, and business optimisation. Although larger businesses have developed a comprehensive framework for implementing DT, considerable ambiguity persists over the pace and efficacy with which Small and Medium Enterprises (SMEs) can adopt this new methodology [36]. The implementation of ML for DTs presents several challenges: data must be effectively integrated, cleaned, and preprocessed; models must adapt to non-stationary conditions, handle missing or corrupted data, and maintain interpretability for engineers and decision-makers. Ultimately, secure, privacy-preserving infrastructures are essential to prevent data leakage or adversarial attacks. Furthermore, CI/CD pipelines in MLOps ensure that models are not only effectively deployed but also adequately maintained, monitored, and updated throughout their lifecycle [21].

In the past decade, ML algorithms have emerged as a potential solution to alleviate the computational burden of conventional optimisation methods and offer near-optimal solutions for highly non-convex problems [37]. Data collected from various network devices (e.g., mobile terminals, access points, IoT devices, edge servers) is transmitted directly to a high-performance computing server for appropriate model training in centralised ML approaches. Afterward, model inference is performed on all relevant devices, if required. Nonetheless, this approach presents some problems, particularly in the contemporary context of IEC systems: (i) Centralised data collection may result in significant computational burden, particularly with numerous devices and datasets, as well as create a single point of failure; (ii) given that the IEC continuum envisions multiple interconnected heterogeneous devices across various infrastructures, data preprocessing prior to the training of the ML model is essential, potentially prolonging overall training duration and increasing system latency. (iii) The regular transmission of data from IoT devices to centralised servers may raise security and privacy issues, as not all IoT devices possess the processing capability to implement sophisticated security protocols; and (iv) resource-intensive ML training could affect the energy consumption of the devices involved [38].

Distributed ML methodologies can reduce the centralised computing load by either parallelising training processes or by effectively distributing training datasets [39, 40]. To address the above-described issue, the idea of FL has arisen in recent years as a promising strategy that facilitates networked ML training while also preserving privacy. Consequently, training is conducted locally on the participating devices, eliminating the necessity of transmitting training data to remote servers [41, 42]. At specific time intervals, the trained model's parameters are transmitted to the central processing node, where the master model is periodically updated. Furthermore,

localising training data enhances privacy, as previously noted. With FL, data can be distributed across multiple devices, enabling the use of significantly larger datasets. The volume of data transfers and the communication load are reduced, particularly when data is dispersed across devices with limited connectivity or bandwidth. Ultimately, FL enables the model to be trained on a variety of data sources, enhancing its resilience, generalisability, and overall training efficiency. The promise of FL is appealing to many users. There are several key advantages to take note of [43]:

- Training time is reduced. Multiple devices are used to calculate gradients in parallel, which offers significant speedups.
- Inference time is reduced. At the end of the day, each device has its own local copy of the model, allowing predictions to be made extremely quickly and without relying on slow queries to the cloud.
- Privacy is preserved. Uploading sensitive information to the cloud poses a significant privacy risk for applications such as healthcare devices. Privacy breaches in these settings may be a matter of life and death. As such, keeping data local helps preserve end users' privacy.
- Collaborative learning is easier. Instead of having to collect a single massive dataset to train an ML model, federated learning enables a"crowdsourcing" approach that can make data collection and labeling much easier, reducing the time and effort required.

Given the natural advantages of both edge computing and FL, and the fact that edge computing is a highly suitable environment for deploying FL, the promising future of edge FL motivates us to propose it in an industrial case study, aiming to enhance the reliability of the proposed framework. In this study, we implemented several custom FL strategies to leverage decentralized training. Generated baseline results using publicly available industrial datasets. The details of the implemented strategies and their baseline results will be discussed in the relevant chapters of the dissertation.

1.2 Research Questions

Despite the significant attention that DTs have received in academia and industry, and the challenges faced in the current study, some research questions are identified, and the whole dissertation addresses them in its subsequent sections.

RQ1: Which ML and DS techniques could be used for industrial datasets for detecting anomalies?

RQ2: How does DT address the growing needs of Industry 4.0?

RQ3: How should the privacy of the industrial dataset be preserved during local ML model training at the edge layer?

RQ4: How to achieve Proof-of-Concept from Off-the-shelf systems in IEC environments?

RQ5: What are the mechanisms to automate, monitor, and trigger the AI models in soft real-time IEC environments?

RQ6: How can AD in smart manufacturing systems be achieved in a time-constrained IEC environment?

RQ7: In the IEC context, how do DTs enable the deployment of AI Models at the edge layer?

RQ8: How does DT leverage MLOps in the IEC environment?

1.3 Research Objectives

Addressing the above research questions, the ultimate goal of this dissertation is to develop an automated, adaptable, and federated DT for IEC environments that uses industrial datasets to train ML models at the cloud/edge when fewer computing resources are available, while ensuring data privacy. If the datasets are large enough, they are transferred to the cloud layer, where Azure AI services train the model using the datasets and send the trained model back to the edge layer for deployment. Docker has handled edge deployment. This is why we said the DT is adaptable: it handles multiple datasets across various scenarios. This DT has been implemented in a specific industry use case, which has verified its credibility and provided a Proof of Concept. A detailed requirements specification and extraction were completed with industry officials before implementation, and industrial Data Analysis using Statistical Process Control (SPC) was then carried out for ML model deployment at the edge layer using custom FL techniques. Continuous Integration and Continuous Deployment (CI/CD) have been implemented using GitHub Actions, which trigger the development of the ML model, extract its artifacts, and deploy it to the edge device. Additionally, the edge manifest is updated in the industry. In this study, GitHub workflows meet the needs of MLOps by storing the latest model version in the GitHub repository. Based on model performance, we can trigger either the latest ML model or a previous one at the edge layer. By default, the latest ML Model must be deployed because it was trained on the newest dataset, but in some scenarios, the previous model can also be deployed.

Another essential objective is to build innovative FL strategies at the edge layer that ensure data privacy during ML model training. Let's summarize our main contributions in the paragraph below. A detailed explanation of the proposed DT's implementation is provided in the following chapters.

1.4 Major Contributions

The main contributions of this dissertation are: Firstly, the dataset of DT machines from an industry has been analyzed using SPC. Basic and advanced statistical operations and techniques are applied to analyze the dataset and identify normal and anomalous processes. The basic steps include finding the mean, average, and standard deviation of a dataset. Finding the dataset distribution using Python's visualization libraries like *seabon*, *matplotlib*, and *pyplot*. Dataset normalization and feature extraction have also been performed as important steps in ML model training. ARIMA and SARIMA analyses have been conducted on the industrial dataset to examine its seasonality and patterns. Custom dataset grouping was performed after selecting the most critical columns for detecting anomalous operations, and then

applying State-of-the-Art (SOTA) unsupervised ML algorithms to identify anomalous behaviour in industrial machines.

Secondly, we developed an industrial DT, **C2E-AIOps**, an IEC-based automated framework that meets the industrial needs of SMEs. C2E-AIOps DT Framework is implemented on a specific industrial use case. i.e., Marposs S.p.A. [44]. The proposed DT is containerised using Docker technology, and also utilises Azure Functions [45]. This serverless function is triggered via an API and automates the AI and MLOps-based implementation of industrial DT. FL has been proposed in this framework at the edge layer, and custom FL strategies have been developed there, yielding initial baseline results. Experimental results show that the proposed DT effectively leverages AI and MLOps best practices in a specific industry and automates the entire implementation workflow, from cloud-based ML model training to model deployment on an edge device. C2E-AIOps is adaptable and can also be implemented in other SMEs with similar workflows, for which our DT is designed.

Finally, to better meet the demands of industrial deployment, we also proposed **FedAdapt-CAD**. A novel FL framework that combines adaptive aggregation with dynamic learning techniques. The adaptive aggregation strategy improves convergence and fairness in non-independent and identically distributed data (non-IID) settings by weighting client updates based on local data quality and model performance. Dynamic server-level aggregation is a hyperparameter-tuning mechanism that ensures model robustness and longevity. This framework improves accuracy, recall, and convergence speed across three real-world industrial datasets —CMAPSS, WADI, and BATADAL —compared with other SOTA FL algorithms (FedAvg, FedProx, and FedH2L).

1.5 Structure of the Dissertation

The following dissertation outline provides a concise overview of the components of this study. The text comprises seven sections, including an introduction to the subject. The report is organized as follows:

- Chapter 1: *Introduction*. It describes the importance of ML algorithms for AD in smart manufacturing applications. Discuss the complete workflow of the AD cycle in the IEC continuum. Includes a description of smart manufacturing significance in Industry 4.0, IEC computing role in DTs, research problems for DTs, its challenges in academia and industry, and hypotheses to address these problems, objectives of this study, brief implementation details, and all the major contributions are described in this section.

- Chapter 2: *Background and Related Work*. This chapter compiles research publications and reports on DTs, ML, and AutoML system frameworks. These documents extend from the year of the topic's inception to the present. They are examined and elaborated to enhance understanding of the current research problem. Research gaps have been identified and analysed deeply to address this dissertation's hypothesis and research questions.

- Chapter 3: *Marposs Dataset Analysis and Implemented ML Techniques*. Describe brief details of the industrial dataset used, dataset visualization techniques, and advanced SPC methods to analyze the industrial data. An anoma-

lous industrial case has been discussed, and the grouping mechanism explains the data preparation steps required to apply clustering techniques to it for anomaly prediction. Performance matrices have been examined to evaluate the efficiency of these clustering techniques.

- Chapter 4: *Research Design and Proposed Methodology.* This chapter describes in detail the conceptual framework of the C2E-AIOps framework. It provides a comprehensive explanation of the development of the IEC-based DT framework, detailing its architecture and the code's functionality. The methodologies employed for preprocessing, model assessment, training, and deployment at the edge layer and on the edge device are examined.

- Chapter 5: *Implementation Details, Results, and Discussion.* An illustration of the developed framework that facilitates the reader's understanding of the framework from a functional perspective. System details, implemented workflow results, CI/CD automation, and prediction results have been evaluated. The outcomes generated by the proposed C2E-AIOps framework are discussed here. It discusses the implementation results of a particular industrial case study.

- Chapter 6: *FedAdapt-CAD.* This chapter discussed the proposed FL framework in detail. Its architecture, workflow, dataset description, and implementation results and performance evaluation have been evaluated comprehensively. Moreover, a critical assessment of the results has been conducted.

- Chapter 7: *Conclusion and Future Work.* In the final chapter of the dissertation, a summary of the completed tasks and achieved objectives is provided. The proposed C2E-AIOps and FedAdapt-CAD frameworks have been evaluated and compared. Future directions have been identified to enhance the proposed system's adaptability in other industrial scenarios.

Chapter 2

Background and Related Work

This chapter discusses previous works related to deploying ML models in industries, AutoML approaches, AI and MLOps practices, and research conducted on the implementation of the DT frameworks in Industry 4.0.

2.1 ML Models Deployment in Industry 4.0

ML-enabled systems that use ML models are already common. Still, there is limited real-world evidence about how ML-enabled systems are built in practice, particularly in terms of sharing ML models. There is currently significant emphasis on defining best practices for the development of ML systems by practitioners experienced in creating large-scale ML systems, such as those at Google [46] and Microsoft [47]. Paleyes et al. [48] say that even if ML is becoming more popular quickly, there is still a big gap between making ML models in testing environments and using them successfully in the real world. The primary objective of ML-based fault detection systems is to detect any malfunctions within the system. In numerous industrial applications, it is essential to identify multiple faults. A prevalent approach is employing a singular multiclass classifier to detect the entire array of problems. There are two primary categories of multiclass classifiers: model-based, such as Support Vector Machines (SVM) [49,50], and Neural Network (NN) based [51]. Adeli and Mazinan [51] found comparable performance using NN to that of achieving a detection quality of over 90 percent. This multiclass notion possesses a significant deficiency, rendering it unsuitable for implementation in Industry 4.0. This limitation pertains to the circumstance that all sensors in the facility support a singular classifier. This fact establishes a centralised framework where data is consolidated at a singular point in the network, contradicting the necessity for decentralisation [52]. de Assis Boldt et al. [53] employed an ML technique utilising an NN known as extreme learning machine (ELM) for sensor selection, necessitating more computer resources than SVM. The following section highlights specific industrial use cases, as our ultimate goal is to propose a framework for a targeted industry first.

The Tennessee Eastman Process (TEP) [54] serves as a prominent benchmark for industrial AD, emulating a realistic chemical production facility equipped with several sensors and actuators regulating the process. It accommodates standard operations and specified fault scenarios, providing labelled datasets that enable reproducible assessment and comparison of fault detection methodologies [55, 56]. Its efficacy is rooted in its ability to replicate authentic process dynamics, correlations, and

non-linearities, rendering it more representative than rudimentary synthetic datasets. From an implementation perspective, TEP's initial Fortran-based simulation necessitates unique configuration or language adaptations for integration with contemporary ML pipelines. The fixed fault set restricts adaptability to new or developing failure modes and fails to consider specific Industry 4.0 elements, including CPS, network latency, and hybrid control logic.

Generally, there are three types of ML techniques: Supervised, Unsupervised, and Reinforcement Learning. Semi-supervised is a mixed type that is also discussed in this domain. Which ML technique could be used in industrial settings? The answer is Subjective because in industrial AD, the majority of real-world datasets are fundamentally unsupervised or semi-supervised, rather than entirely supervised. This is due to the extreme scarcity of labelled fault data in operational settings. Faults are rare, and when they do occur, they typically result in expensive downtime, rendering deliberate fault injection for data acquisition impractical. Furthermore, fault labelling in older records may be deficient or erroneous due to human error or restricted sensor coverage. Consequently, much of the accessible data reflects standard operating conditions, so promoting the application of semi-supervised methodologies in which models are trained solely on typical patterns, with abnormalities recognised as deviations. Comprehensively supervised industrial datasets—featuring balanced and accurately labelled instances of normal and fault conditions—are scarce and generally derived from simulations or controlled experiments, such as benchmark datasets (e.g., TEP, SWaT, WADI). Thus, whereas supervised approaches are useful for benchmarking, practical applications frequently depend on unsupervised or semi-supervised techniques to address the absence of dependable fault labels. Therefore, we mainly addressed the unsupervised and semi-supervised work here.

2.2 Unsupervised and Semi-supervised ML Use Cases

This section also addresses the ML approaches for specific industrial use cases. The aim is to understand which ML algorithms are suitable for the particular industrial use cases, and for our industrial use case, which ML algorithm could be deployed.

This study [57] introduces a semi-supervised ML approach for fault detection and diagnosis in rooftop Heating, Ventilation, and Air-Conditioning (HVAC) units. It uses unsupervised feature extraction techniques to learn system behaviour from unlabeled data, then incorporates limited labelled fault samples to refine the classifier. This method reduces the need for extensive fault labelling, which is costly and time-consuming in industrial settings. The approach improves accuracy in detecting fault types while maintaining robustness to data imbalance. However, it faces limitations such as relying on quality unlabeled data, balancing labelled and unlabeled samples, and limiting adaptability to emerging fault types. Gao Huang et al. [50]introduce semi-supervised and unsupervised Extreme Learning Machines (ELMs) for classification tasks with scarce labelled data. The semi-supervised Extreme Learning Machine (SS-ELM) utilises graph-based manifold regularisation to exploit unlabeled data by capturing the inherent data structure, while preserving the computational efficiency typical of ELMs. Despite these benefits, the efficacy of SS-ELM is heavily dependent upon the quality of the similarity graph and necessitates meticulous parameter

optimisation, potentially constraining robustness and generalisation. The static random hidden layer of ELMs may limit adaptability to intricate or dynamic industrial data, and the model's opaque nature presents difficulties for interpretability. These constraints indicate significant research deficiencies in adaptive graph building, improved model interpretability, and versatile architectures to accommodate dynamic industrial fault detection contexts more effectively.

A lot of recent research has been conducted on how ML might help solve the problems and challenges that IoT systems face. Samie et al. [58]provided an overview of ML algorithms and their applications. They assessed the incorporation of ML in several IoT domains, classifying their functions and identifying their position on the cloud-to-edge spectrum. Furthermore, they examined the existing challenges and research opportunities for effective ML in the IoT. They thoroughly analysed papers on ML and the Internet of Things, categorising them according to the ML approach employed. Usama et al. [59] centred on recent advancements in unsupervised ML and their application inside the networking domain. They incorporated IoT, software-defined networks (SDN), and self-organising networks (SON) as prospective network architectures. Bangui et al. [60] examine big data clustering algorithms for IoT applications, focusing on their ability to handle the volume, velocity, and variety of IoT data streams. They review traditional methods like k-means and hierarchical clustering, as well as more scalable algorithms for IoT environments. However, the study highlights limitations such as a lack of scalability for high-dimensional, high-velocity IoT data and the assumption of static data distributions. The authors suggest the need for adaptive, lightweight clustering methods, real-time processing, data heterogeneity handling, and real-world deployment studies. Tsai et al. [61] highlight the various methodologies used for AD in IoT, including signature-based and semantic-based methods, which utilise statistical, machine, or protocol-specific information.

Bernoulli Random Forests [62] discuss the basics of Random Forest and meaningfully bridge theoretical consistency and practical performance, providing robustness across classification and regression tasks. However, for industrial applications, their practical adaptability may be hindered by additional hyperparameters, computational overhead, limited robustness under data drift, integration challenges, and interpretability concerns. [59] talks about partitional clustering in general and then talks about the benefits of this method. They begin by saying that clustering is a way to group data into distinct groups. Partitional clustering is better than other forms of AD algorithms since it may use information about the size of clusters by using the provided distance functions. These functions make sure that the many types of clustering algorithms provide data shapes that are correct. [63]says that partitional clustering has a number of problems. The problems are determining the number of clusters, the impact of outliers and data noise, and the problem of starting clusters. Qi et al. [64] propose k*-means, an improved K-means variant that improves clustering stability and speed through three strategies: hierarchical initialisation, incremental updates, and pruning of distant clusters. Experimental results show higher accuracy and lower computational cost compared to standard K-means. However, the approach requires tuning hyperparameters, may incur overhead for large datasets, and assumes spatially coherent clusters, limiting adaptability in high-dimensional, irregular, or dynamic data scenarios. This article [65] introduces a two-stage pipeline for identifying anomalies in industrial settings, employing a

hybrid ensemble of three unsupervised models—Local Outlier Factor, One-Class SVM, and an Autoencoder—integrated via a weighted averaging method to enhance precision. The technique, evaluated using data from three air-blowing devices, got superior F1-scores compared to individual models, thereby illustrating its efficacy for real-time monitoring. The dependence on constant combination weights restricts adaptability to fluctuating operational settings, and the unsupervised design may inadequately identify minor or emerging anomalies without regular retraining or dynamic modification.

Applying unsupervised ML techniques is subjective based on a specific industrial dataset and its manufacturing processes, as it was highlighted by many researchers in the literature. Considering these factors and understanding our particular industrial dataset, we implemented clustering techniques K-Means, DBSCAN, and Isolation Forest. They effectively work, and their deployment details are in Chapter 3. Implementing just ML models on an industrial dataset is not enough, which is why we researched some Automated ML frameworks that could be deployed in an IEC environment.

2.3 Automated ML-based Frameworks

This section explores AutoML-based frameworks for specific research problems. Over the past decade, ML research has seen a surge in algorithms, but their performance is highly dependent on various factors, requiring significant human effort. Automated machine learning (AutoML) aims to make these algorithms more accessible for researchers of varying expertise levels. In this section, we discussed some supervised and unsupervised automated approaches for AD in industries. An overview of the AutoML pipeline is in Figure 2.1 [66].

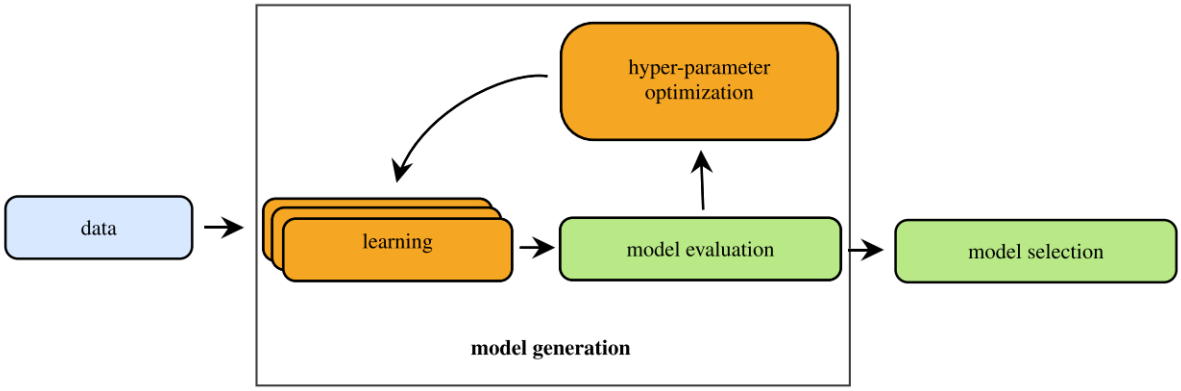

Figure 2.1: Overview of AutoML pipeline

Researchers produced multiple solutions and tools for the autoML challenge, including Auto-Weka [67], [68,69], an open-source solution developed in Java utilising Weka ML software. It employs Bayesian optimisation and SMAC [70] to explore classification and regression techniques as well as hyperparameter configurations. Nonetheless, auto-Weka has not issued a new version since 2017 and is intended for supervised learning, omitting modern NN. Likewise, auto-sklearn [71], [72] has been developed in Python utilising the widely-used scikit-learn framework. Recently, auto-sklearn 2.0 has been introduced to enhance the effectiveness of auto-sklearn by utilising

portfolios via meta-learning. It employs analogous optimisation methodologies to auto-Weka, aiming to improve the optimisation process and to initiate Bayesian optimisation by recognising analogous datasets and leveraging previously acquired information. Like auto-Weka, this framework lacks functionality for deep neural networks and unsupervised algorithms. Hyperopt-Sklearn [73] is an alternative automated ML framework that utilises the scikit-learn library. This open-source framework employs hyperopt [74], which facilitates various optimisation approaches, to delineate a search space for the potential configurations of scikit-learn learning algorithms. Under the presumption that the complete dataset is available prior to initiating the procedure, many techniques have been offered in static or batch environments.

Currently, AutoML is entirely devoted to supervised learning, although some strategies might work for unsupervised activities. This section aims to review the proposed research projects for the clustering job, where the model selection and hyperparameter tuning should be done without prior knowledge of data labels. Calculating the degree of similarity between cases in the same cluster and the degree of dissimilarity between instances in different clusters is a necessary step in evaluating clustering strategies. Similar to the supervised task, there aren't many attempts to improve performance in the clustering task by using ensemble and meta-learning (also called meta-clustering) techniques [75–77].

In the stream framework, a new technique for automatically configuring clustering algorithms for changing data streams has been presented, namely ConfStream [78]. ConfStream also employs an ensemble of a variety of stream clustering algorithms with various configurations, which were inspired by the Blast approach [79]. The best-performing configuration will be utilised to generate a new one that will replace one of the ensemble's configurations (if it is good enough) based on the evaluation of each configuration and algorithm. Carnein et al. assessed each configuration's clustering quality using the silhouette width metric. As a result, the configuration with the highest cluster quality is chosen for the stream's subsequent window. But like the Blast approach [79], ConfStream uses a tiny portion of the stream (window) on a regular basis to choose the model and configuration that performs the best, then uses them for the next window of the stream, and so on. ConfStream is a very promising approach for unsupervised AutoML for data streams, albeit using a batch-incremental paradigm. confStream enhances automatic stream clustering but has problems in real-time scalability, generalisation to unfamiliar data situations, and rapid adaptation to idea drift in industrial or large-scale streaming contexts.

A system for outlier identification, termed PyOD, has been developed in [80] as a Python toolkit for scalable AD, incorporating over 20 detectors with established outlier ensembles and network-based methodologies. However, this methodology does not address issues such as optimal pipeline design, namely the search for optimal hyperparameter settings. An automated Python framework for Outlier Detection with Database Support (PyODDS) has been introduced in [81] to automatically optimise the detection pipeline for novel datasets. It provides access to a diverse array of anomaly detection algorithms. PyODDS examines a new dataset to identify an optimal method and its configuration inside a specified search space, which encompasses a variety of hyperparameter settings. The primary limitation of this approach is that, although unsupervised AD techniques are employed in the search space, the evaluation of various algorithms and configurations is conducted using the

F1-score, a supervised measure. This system fails to tackle the cold-start problem, as it merely initiates with the default configuration for each member and is unsuitable for dynamic data streams due to its reliance on static algorithms. The Locally Selective Combination in Parallel Outlier Ensembles (LSCP) framework [82] is a method for unsupervised AD. It uses a variety of detection algorithms initialised with hyper-parameters, trained on a training set of instances and derives an outlier score matrix. The local region is defined by the nearest training instances in randomly selected features of t sub-spaces. The best performing detector is identified by measuring the similarity between base detector scores and pseudo ground truth. However, LSCP has limitations such as using the same base algorithm with different configurations, performance degradation when irrelevant attributes are present in high-dimensional spaces, and high time complexity due to the nearest neighbour search.

The Automated Time Series Outlier Detection System (TODS) is an automated framework [83] for AD on time series data. It offers data processing, feature extraction, and detection algorithms. The system is user-friendly and provides the most suitable algorithm based on F1-score, but lacks a precise search technique for pipeline selection. Zhao et al. [84]have developed MetaOD, a meta-learning approach for unsupervised outlier detection. This method uses the past performance of detection models on existing benchmark datasets to automatically select an efficient model for a new dataset. Meta-features capture statistical properties of data distributions, and the method identifies datasets with similar characteristics to the new one. It then focuses on models that perform well on those datasets, similar to a recommendation task. The method discretises the hyper-parameter space, presents models as model, configuration pairs, and evaluates models using a supervised measure, making MetaOD not fully unsupervised. Other meta-learning methods, like Instance-Specific Algorithm Configuration (ISAC) [85], cluster meta-train databases [86, 87]and select the best model based on new data. However, these techniques are limited to one model class and are not applicable to choosing among different heterogeneous algorithms.

In [88], Zimek et al. addressed the fundamental difficulty and reviewed several AD systems, primarily those based on approximate neighbourhoods, that manage such data. A recursive binning and re-projection method has been presented in [89] that initiates by recursively partitioning the data into k clusters, employing a clustering technique such as k-means. The data within each of the k bins is further clustered into k bins, continuing this process until a bin lacks sufficient instances. Subsequently, for each resultant bin, the neighbouring elements are enumerated, and the authors employed a nested loop to extract the n outliers. The authors posited that the quantity of abnormalities n is an input parameter, which is typically not applicable for unsupervised anomaly detection systems. They also conducted tests with a singular number; nonetheless, it would have been preferable to experiment with a range of alternative values of n to ensure consistency in the results. While this study offers important algorithmic advancements for outlier mining in high-dimensional spaces, fundamental challenges related to high-dimensional data properties, parameter tuning, and dynamic data remain open for further research.

We have proposed an Azure ML-based AutoML Service that is a sort of no-code implementation and very easy to use by non-coding professionals or industry stakeholders. AutoML works on both supervised and Unsupervised ML problems, making it adaptable to different industrial use cases. It is a cost-effective and

pipeline-based service that helps record and deploy ML models in IEC settings. Its complete implementation is discussed in Chapter 4.

2.4 MLOps based Approaches

This section highlights the MLOps deployment at Edge and Cloud layers for some specific industrial use cases. Raj et al. [90] present an Edge MLOps framework designed to automate the continuous training, deployment, delivery, and monitoring of ML models in edge and cloud environments, validated through a private campus air quality forecasting case study. The framework exhibits functional stability and automated retraining in response to performance decline. Its applicability is constrained by validation conducted in a confined, controlled setting. There are still too many unanswered questions about how to make this work on a large scale with different types of hardware and platforms besides Azure. How to handle changes in workload in the real world. Although it presents a potential framework for Edge MLOps in AIoT, numerous significant real-world and industrialisation difficulties remain inadequately investigated.

Sato [91] presents MLOps as an adaptation of DevOps ideas for ML, including recommended practices to optimize the ML lifecycle from development through deployment and monitoring. It offers significant conceptual direction and practical recommendations, but the content is predominantly high-level and lacks empirical evaluation and comprehensive implementation roadmaps. The scope concentrates on Google Cloud's ecosystem, restricting discourse on cross-platform compatibility and relevance in diverse infrastructure environments. This paper provides an approachable introduction to MLOps practices, but does not address numerous technical, operational, and scalability aspects. Tamburri's [92] study investigates sustainable MLOps, highlighting emerging trends and significant hurdles in the development of ecologically and operationally sustainable ML pipelines. The paper presents a timely and forward-looking discussion; nonetheless, it is predominantly conceptual and exploratory, lacking empirical validation and case studies. The discourse on sustainability is extensive, although it lacks specificity about particular measurements, approaches, or instruments for quantifying and mitigating the environmental impact of ML workflows. Moreover, issues including scalability, cross-platform interoperability, and integration into current DevOps or data engineering frameworks are only superficially addressed. This study offers significant conceptual framing and a research agenda for sustainable MLOps; however, it defers practical implementation tactics and extensive validation.

Wu et al. [93] discuss Facebook's ML inference methodology, focusing on system topologies, optimization techniques, and real-world applications. However, the research is limited to Facebook's internal ecosystem. It focuses on technological performance and system optimization, but lacks comprehensive benchmarks or open-source resources. The report provides a comprehensive industrial viewpoint but defers further examination of portability, reproducibility, and cross-domain application to future research. Lee et al. [94] examine model serving in ML.NET, emphasizing the facilitation of ML inference from edge devices to cloud environments via a cohesive framework. The work offers valuable architectural insights and underlines the adaptability to ML.NET for cross-environment deployment. However, it is predominantly descriptive and lacks comprehensive experimental validation or substantial

performance testing. The study fails to thoroughly investigate issues such as latency optimization on heterogeneous networks, model versioning in remote environments during model transmission and execution. The considerations of integration with non-Microsoft ecosystems, economic viability, and long-term sustainability are constrained. This study provides a realistic architectural viewpoint, but has significant concerns about scalability, interoperability, and operational resilience for future investigations.

Agrawal and Rawat [95] examine DevOps as a contemporary methodology for cloud development and testing, highlighting its capacity to enhance collaboration, automation, and deployment efficacy. The study is predominantly conceptual, offering a broad overview without comprehensive implementation case studies. The discourse fails to adequately address the issues of integrating DevOps with large-scale cloud infrastructures. The report also provides minimal focus on toolchain compatibility, scalability with fluctuating workloads, and organizational change management, so it defers numerous practical adoption and operational issues to future research.

Ereth [96] presents a conceptual description of DataOps, aiming to formalize its concepts, procedures, and functions to enhance the agility and reliability of data analytics pipelines. Although the study offers a significant theoretical framework and elucidates essential terminology, it is predominantly descriptive, lacking empirical validation, case studies, or quantitative performance assessment. The discourse fails to thoroughly examine the technological obstacles of executing DataOps in diverse, large-scale, or real-time data contexts, nor does it extensively investigate its interaction with associated methodologies like DevOps or MLOps. Consequently, the study functions as a significant conceptual foundation; however, it did not address numerous practical implementation and evaluation considerations for subsequent research. Munappy et al. [97] explore the transition from ad-hoc data analytics to systematic DataOps frameworks, highlighting the benefits of collaboration, automation, and continuous delivery. It fails to address technical obstacles like toolchain integration, scalability for high-velocity data, and real-time data quality assurance. The study provides a conceptual contribution but leaves precise technical and operational viability considerations to future research.

We proposed GitHub Actions, a very cost-effective and easy-to-deploy MLOps approach that was implemented in a specific industry case, Marposs S.p.A. [44]. This approach was implemented in a complete IEC setting. This pipeline approach is scalable, adaptable, and operational, and can be implemented on other industrial use cases. The full implementation details are in Chapter 4.

2.5 DT Frameworks

This section offers a thorough examination of the current research and developments in DTs in industrial settings, especially within the manufacturing sector. It examines the problems faced in the production of DTs. It delineates the constraints linked to offline legacy systems, the absence of cloud-based infrastructure support, the necessity for bi-directional communication, and the characterisation of consistency between the physical twin and the digital twin. This part seeks to provide a robust foundation for the proposed research by reviewing pertinent literature, with a focus on creating an innovative DT infrastructure. This literature review part critically analyses existing knowledge on remote monitoring and control capabilities, incorporating a

real-time delay characterisation technique. It highlights the potential influence of proposed research on improving efficiency, accuracy, collaboration, and quality in manufacturing and assembly systems.

Tao et al. [98] suggested using DTs in product design, manufacturing, and service to optimize smart manufacturing systems. They emphasized the potential of DTs to streamline processes and improve operational outcomes throughout the product lifecycle. This study [98] lacks experimental validation on industrial-scale case studies. It does not address interoperability, real-time data processing, scalability, and human and organizational factors. The work serves as a visionary blueprint but leaves significant technical, operational, and implementation questions unanswered. Lu et al [99] provides a comprehensive overview of DT-driven smart manufacturing, but lacks empirical validation and detailed implementation cases. The study lacks strategies for interoperability, large-scale real-time data processing, and integration with legacy manufacturing infrastructure. Despite providing a strong theoretical foundation, many practical, technical, and operational considerations remain open for further investigation.

This study [100] presents an open-source approach for designing and implementing DTs in smart manufacturing. However, the study is limited-scale and lacks validation in large, complex, or high-throughput industrial environments. The model does not address challenges in integrating with diverse systems, real-time data synchronization, long-term maintenance, and organizational readiness. This leaves significant practical, security, and large-scale deployment issues for future research. [101] propose a DT-based framework for the management and control of smart manufacturing in complex product assembly environments, emphasizing its capacity to enhance coordination, monitoring, and decision-making. Nonetheless, the framework is mainly supported by conceptual modeling and a restricted case demonstration, lacking extensive industrial trials or quantitative performance assessment. The study inadequately addresses compatibility with diverse equipment, issues in real-time extensive data processing, and integration with current factory execution systems. The research presents a potential architecture for complicated assembly management; yet, it raises significant technical, operational, and adoption-related inquiries for future investigation. Leng et al. [102] present a DT-driven methodology for the swift reconfiguration of automated manufacturing systems through an open architecture concept, facilitating flexibility and adaptation in dynamic production settings. Although a prototype demonstration endorses the idea, it lacks validation in high-complexity industrial environments, resulting in insufficient exploration of scalability and robustness. The study insufficently addresses difficulties concerning compatibility with various industrial systems and real-time data synchronisation. Conclusively, the study presents a new reconfiguration framework while delaying essential technical and operational aspects for subsequent exploration.

Jiang et al. [103] advocate for a DT methodology to augment virtual–real integration in Industrial IoT, to enhance monitoring, control, and decision-making functionalities. The paper presents a clear conceptual framework and outlines prospective benefits. However, its validation is confined to a theoretical model and illustrative examples, missing extensive industry case studies. The study fails to tackle issues like interoperability among diverse IIoT devices thoroughly and real-time synchronization amidst substantial data volumes. The paper offers a key conceptual framework for the integration of IIoT and DTs; however, it leaves considerable

practical, technological, and scalability challenges for future investigation. Hung et al. [104] present a DT framework for intelligent manufacturing, utilizing container technology and cloud services to enhance modularity, scalability, and deployment flexibility. However, the methodology's effectiveness in diverse industrial settings is uncertain due to limited validation under controlled conditions. Issues such as real-time synchronization, latency control, and integration of older systems are not addressed. Further empirical assessment is needed to validate its robustness and interoperability.

Sudharsan, Breslin, and Ali [105] introduce Edge2Train, a framework designed for training ML models, particularly SVMs, directly on resource-limited IoT edge devices to diminish dependence on cloud-based training. Although the approach exhibits viability via prototype implementation, the evaluation is constrained to particular model types and restricted datasets, resulting in ambiguity regarding performance with more intricate algorithms or bigger, real-world workloads. The framework's adaptation to diverse hardware, energy management, and robustness under variable network conditions is inadequately handled. Whereas Edge2Train presents a promising avenue for on-device learning in IoT environments, substantial issues regarding scalability, generalizability, and operational efficiency persist for future investigation. Moon, Kum, and Lee [106] developed an IoT data analysis platform that uses edge-cloud collaboration to predict indoor PM10 and PM2.5 air quality levels. However, the platform's validation is limited to a specific environmental monitoring scenario, raising concerns about its applicability to larger IoT deployments. The study also fails to thoroughly examine scalability, interoperability, network resilience, real-time latency assurances, data confidentiality, security, and long-term operational expenditures.

Bhattacharjee et al. [107] introduced Stratum, a BigData-as-a-Service platform, designed for the lifecycle management of IoT analytics applications. It streamlines deployment, scaling, and maintenance in dispersed settings. However, its effectiveness in large, heterogeneous, and resource-limited IoT ecosystems is uncertain due to its limited scale and diversity. The framework fails to address interoperability with diverse data sources and optimisation among variable workloads and network conditions. The discussion on long-term maintainability and integration with upcoming MLOps or DataOps methods is limited.

Our proposed C2E-AIOps framework presents a complete IEC Continuum-based DT, applicable to a real-world industrial use case (MARPOSS S.p.A.), and leverages the latest cloud technologies, such as Azure ML Studio and AutoML. We addressed interoperability issues in deploying models at the edge by using the ONNX standard for ML models. Our framework is automated, adaptable, and scalable because we have leveraged CI/CD pipelines and a lightweight ONNX model that can be deployed on various hardware and cross-platform devices.

2.6 Contribution Scope and Beyond State-of-the-Art

In the context of the above discussion, this dissertation aims to build an automated, adaptable, interoperable, and privacy preserved industrial DT framework that leverages AI and MLOps best practices. At first, C2E-AIOps framework was

implemented for a specific industrial use case at Marposs S.p.A. This framework trains ML models in the Azure Cloud using the AutoML service. For deployment and inference on an industrial edge device, this trained model was converted to the ONNX format. We made this inference containerised so it can be deployed to other industrial devices. Using GitHub Actions, we implemented CI/CD for the ONNX model on the industrial edge device. Secondly, our FedAdapt-CAD framework addresses the challenges posed by non-IID datasets in diverse industrial scenarios. It is a sophisticated FL framework tailored for AD in industrial and CPS. The framework presents two innovative mechanisms: CAA, which intelligently allocates dynamic weights to client updates based on data quality and local model performance to improve convergence and fairness in non-IID conditions; and DMA, a server-side optimisation strategy that continuously adjusts global hyperparameters in response to changing data distributions and attack behaviours. FedAdapt-CAD achieves higher accuracy, robustness, and adaptability than traditional FL baselines such as FedAvg, FedProx, and FedH2L. It does this by using benchmark datasets including WADI, CMAPSS, and BATADAL. It provides an initial study and baseline results that can be implemented across diverse industrial use cases. In Chapter 3, we implemented advanced SPC techniques and SOTA unsupervised algorithms on industrial DT machines to find anomalies. In chapters 4 to 6, we discussed the proposed frameworks (C2E-AIOps and FedAdapt-CAD) in detail. These contributions open the way to implement new AI technologies in Industry 4.0 to automate industrial processes for AD and to ensure data privacy during AI model training and deployment across diverse industrial scenarios.

Chapter 3

Marposs' Dataset Analysis and Implemented ML Techniques

This chapter discusses the structure of the MARPOSS S.p.A. dataset, its features, and the exploratory data analysis carried out to identify anomalies. It presents various visualization techniques for understanding the patterns of anomalies across different industrial machines and their corresponding tools. Historical analysis was also carried out using ARIMA and SARIMA analysis. A custom grouping method is applied to the industrial anomalous use case to predict the anomalies. Additionally, unsupervised ML techniques were implemented to predict and visualize anomalies in the industrial use case. These implementations highlight the significance and effectiveness of unsupervised ML techniques on industrial datasets. The performance indices also validate the effectiveness of these algorithms in industrial scenarios.

3.1 Dataset Description

Industrial datasets are vast and diverse, often collected from various locations within industrial operations. They can be categorized into three main groups: production data, management data, and external data. Our study and analysis are based on the production data, which includes data from sensors, instruments, and equipment that monitor manufacturing lines, materials, and products. This study takes into account real-time data. The dataset collected over six months from MARPOSS Artis products (Genior [108] and C-thru [109]) installed on a MARPOSS customer's production line was used. The dataset is available as a PostgreSQL database, hereinafter referred to as the "MAINDO database". MAINDO is the MARPOSS product that enables connection and data collection from MARPOSS IIoT products for use with cloud services. Structured Query Language (SQL) queries are used to extract data and store it in a tabular format (such as csv). Initially, we focused on a particular anomaly use case documented in the MAINDO database. MARPOSS Artis' products monitor machine tool operations by observing data from various sensors. The specific anomaly detected concerns operations performed on the "BLOCK" or "HEAD" parts of a given machine. The dataset's attributes are illustrated in Figure 3.1.

This is a time-series dataset. These attributes represent a complete machine cycle. The *machine* column represents the Marposs machine for which historical data has been recorded. The recorded values in the *value* column are based on the sensor signal(Z-axis), as indicated in the *signal* column, because this machine's

	machine	signal	processnumber	toolnumber	dn	bn	indicator	value	start_ts	end_ts
0	MachineB	Z1	9999	0	5	5	FP: Ripple	0.033054	24:13.9	24:32.6
1	MachineB	Z1	9999	0	5	5	FP: Deviation	2.529593	24:13.9	24:32.6
2	MachineB	Z1	9999	0	5	5	FP: Area	2.041341	24:13.9	24:32.6
3	MachineB	Z1	9999	0	5	5	FP: Minimum	-5.039928	24:13.9	24:32.6
4	MachineB	Z1	9999	0	5	5	FP: Maximum	6.387899	24:13.9	24:32.6
...	...	...	...	...	...	...	...	...	...	...
520	MachineB	Z1	9999	0	5	5	FP: Area	1.981314	13:04.7	13:23.4
521	MachineB	Z1	9999	0	5	5	FP: Ripple	0.028309	13:04.7	13:23.4
522	MachineB	Z1	9999	0	5	5	FP: Deviation	2.485922	13:04.7	13:23.4
523	MachineB	Z1	9999	0	5	5	FP: Minimum	-5.919160	13:04.7	13:23.4
524	MachineB	Z1	9999	0	5	5	FP: Maximum	6.958143	13:04.7	13:23.4

525 rows × 10 columns

Figure 3.1: Marposs' dataset overview

primary manufacturing operation is performed on this sensor signal. Other signals are X1, Y1, CSI, IBIS_ACC-1, but we consider only Z1 in the specific machine operation. *toolnumber*, *dn*, and *bn* are fixed indicators and play no significant role. They represent particular parts of the machine related to the current machine cycle. *processnumber* is vital, as we consider a fixed process —let's say 9999 —for the entire machine cycle. This process could be packaging, fabricating, washing, etc. The most essential attributes of this dataset are *start_ts*, and *end_ts*. The machine cycle is operated for a fixed time duration, and at each second, the state of the operation is recorded in the *value* column. *indicator* column conveys the point of the operation, whether it is ripple, minimum point, maximum point, or an area curve of a specific machine operation. The anomalous case recorded by filtering the *indicator* column on FP: Ripple value. We also tried other values, such as FP: Area and FP: Deviation, but found anomalies in the FP: Ripple values. The anomalous case schema is shown in Figure 3.2, which has 105 rows and 10 columns.

3.1.1 Dataset Pre-processing and Visualizations

We apply basic DS techniques to transform data into meaningful insights so ML algorithms can learn from them. It is a prerequisite step for ML. As seen in Table 3.1, *start_ts*, and *end_ts* are actually timestamped column but when we check them originally it gives that these are object type. So, firstly, we convert these objects into timestamp format columns using *to_datetime*() function of Python's pandas library. Afterwards, we checked the mean, standard deviation, min, max, and count of the dataset using the *describe*() function. These basic functions are available in the pandas library. Checking for null or empty values in the dataset is an initial step before processing, and we verify them using the *isnull*() function. In our dataset, there are no null values. We used Seaborn, Matplotlib, and Plotly to analyze the distribution of our dataset. We extracted the skewness, *skew*(), and kurtosis, *kurt*(), of the dataset. These are important function that gives more details about the dataset distribution. In our case, both functions have negative values. It means that

Column	DataType
machine	object
signal	object
processnumber	int64
toolbumber	int64
dn	int64
bn	int64
indicator	object
value	float64
start_ts	object
end_ts	object

Table 3.1: Datatypes of Marposs' Dataset

	machine	signal	processnumber	toolnumber	dn	bn	indicator	value	start_ts	end_ts
0	MachineB	Z1	9999	0	5	5	FP: Ripple	0.033054	24:13.9	24:32.6
6	MachineB	Z1	9999	0	5	5	FP: Ripple	0.033657	40:33.6	40:52.3
11	MachineB	Z1	9999	0	5	5	FP: Ripple	0.032887	58:15.1	58:33.9
16	MachineB	Z1	9999	0	5	5	FP: Ripple	0.032290	37:26.4	37:45.1
21	MachineB	Z1	9999	0	5	5	FP: Ripple	0.032808	00:34.2	00:53.0
...	...	...	...	...	...	...	...	...	...	...
502	MachineB	Z1	9999	0	5	5	FP: Ripple	0.025352	10:42.9	11:01.7
508	MachineB	Z1	9999	0	5	5	FP: Ripple	0.026758	29:50.2	30:08.9
511	MachineB	Z1	9999	0	5	5	FP: Ripple	0.026721	17:16.0	17:34.7
516	MachineB	Z1	9999	0	5	5	FP: Ripple	0.027903	22:24.6	22:43.3
521	MachineB	Z1	9999	0	5	5	FP: Ripple	0.028309	13:04.7	13:23.4

105 rows × 10 columns

Figure 3.2: Marposs' anomalous use case schema

the dataset values are not evenly distributed. Additionally, the dataset's standard deviation is close to 1, indicating significant fluctuations. We also used boxplot visualization to identify outliers in the dataset. Different Industrial DT machines have been visualized for the detection of anomalies. In these visualizations, the timestamp columns are plotted on the x-axis, while the *value* column is on the y-axis. This is because, at each timestamp, based on other indicators in the dataset, the measurement value is recorded in the *value* column, which is why we consider these attributes for visualization. Other attributes are fixed during the machine cycle. In the second attempt, we also visualize the effect of using different tool numbers. We plot *toolnumber* vs *value* and identify which tools have outliers and which do not. The MachineB visualization is shown in Figure 3.3. Similarly, another MachineA has been used to visualize the distribution of the data using boxplots, illustrated in Figure 3.4:

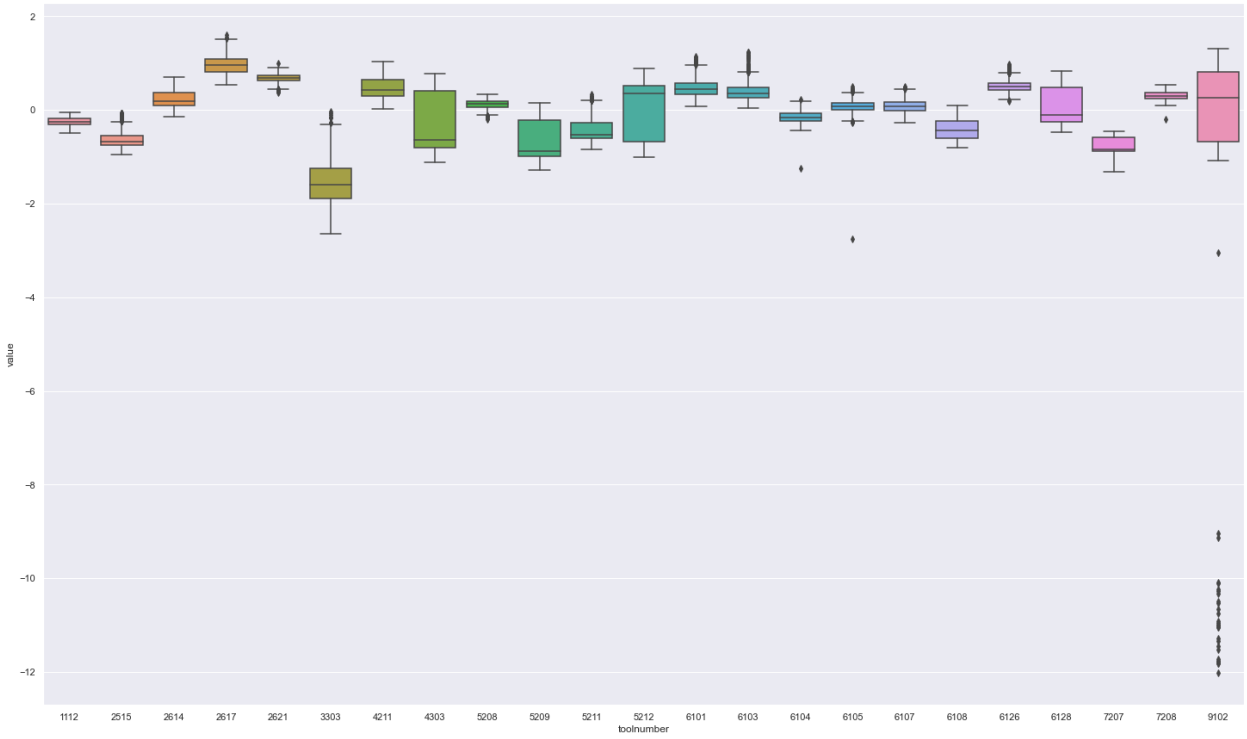

Figure 3.3: Box plot visualisation of MachineB (toolnumber vs. value)

From Figures 3.3 and 3.4, it can be seen that tool number 9102 has outliers, while the other tool numbers are aligned within the -2 to +2 range. This indicates an issue with tool number 9102. Additionally, we also checked the visualization by changing the process number. In this visualization, outlier values for all tool numbers are highlighted in a single graph using two different colors. In a single plot, we can identify which process and which tool have values that may be anomalous and outside the normal range. This is illustrated in Figure 3.4. The blue candles in the figure present process 1, and the orange candle highlights process 11. The purpose of these visualizations is to understand the effect of different processes and tool numbers in an industrial manufacturing machine cycle, identifying both anomalous and normal points. In Figure 3.4, at tool number 9102, process 1 shows more anomalous behavior than process 11. It means that process 1 with tool 9102 has more outlier points than process 11. The same scenario is also interpreted with other tools. Other tools also

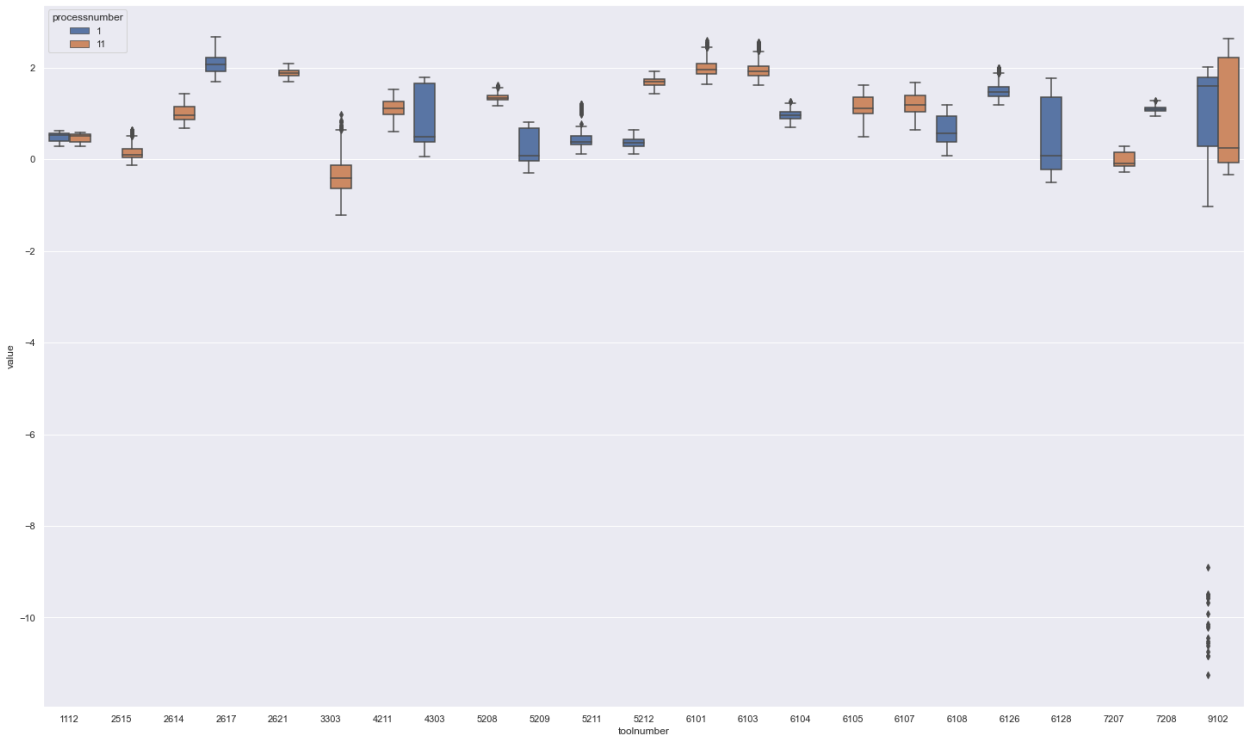

Figure 3.4: Box plot visualisation of MachineA(toolnumber vs. value)

display some anomalous values for processes 1 and 11, but tool 9102 shows more. In this scenario, it can be summarised that tool number 9102 exhibits anomalies when used with process number 1. Also, for MachineA, the boxplot shows more anomalies in the tools. It can be said that the tools associated with MachineA are more anomalous than those associated with MachineB.

3.1.2 Seasonality Analysis

As the industrial dataset spans six months, Autoregressive Integrated Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving Average (SARIMA) models have been applied to it. Both approaches utilize historical data to identify trends and predict future outcomes. The ARIMA model is a widely used method for analyzing and forecasting non-seasonal time series data. It comprises three essential components: Autoregressive (AR), which utilizes historical values to anticipate future ones; Integrated (I), which entails differencing the series to achieve stationarity; and Moving Average (MA), which employs prior forecast errors for predictions. The model is conventionally represented as ARIMA(p, d, q), where p signifies the order of the autoregressive component, d indicates the degree of differencing, and q denotes the order of the moving average component.

The SARIMA (Seasonal ARIMA) model is an enhancement of ARIMA, formulated explicitly for time series data that exhibit seasonal characteristics. It integrates supplementary seasonal elements—Seasonal Autoregressive (SAR), Seasonal Differencing (SI), and Seasonal Moving Average (SMA)—to identify patterns that recur over a specified duration. SARIMA is denoted as: SARIMA$(p, d, q)(P, D, Q)m$, where (p, d, q) denote the non-seasonal parameters, (P, D, Q) signify the seasonal parameters, and m indicates the number of time steps each season (e.g., 12 for

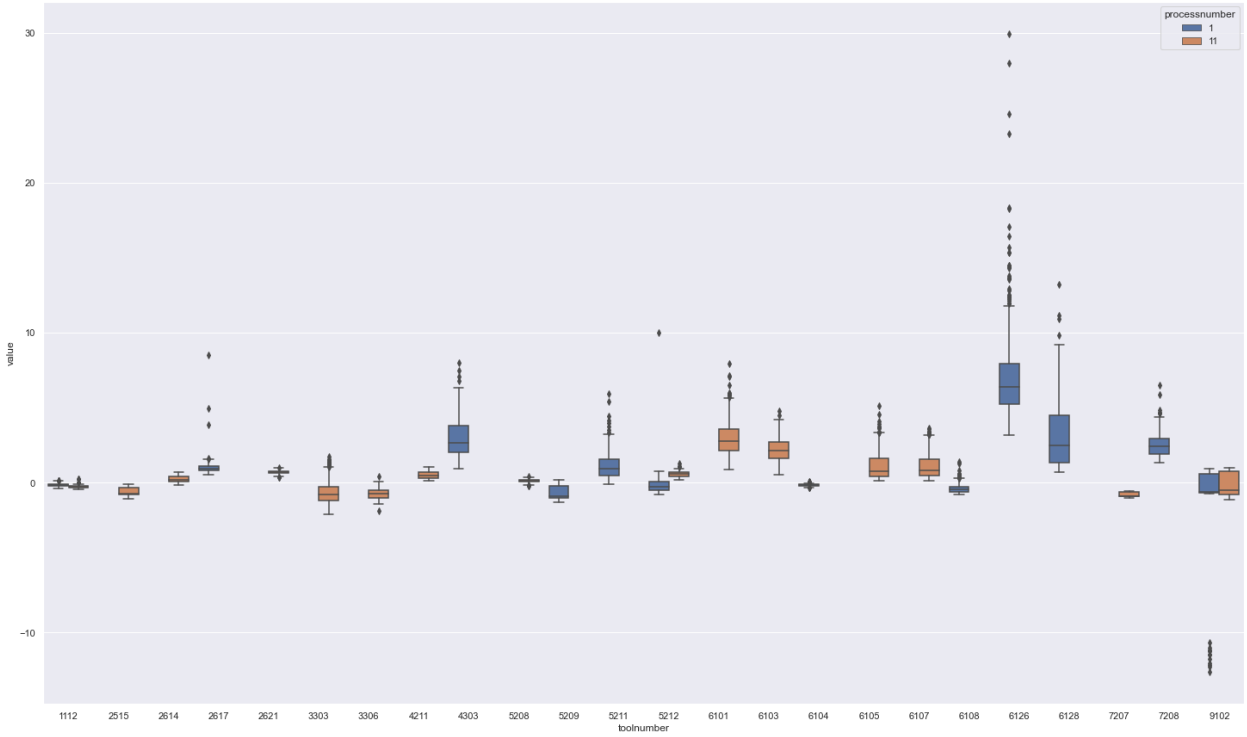

Figure 3.5: Box plot visualisation of MachineA with process numbers 1 and 11

monthly data exhibiting yearly seasonality).

The primary difference between the two models is their appropriateness for various data kinds. ARIMA is suitable for non-seasonal time series, but SARIMA is more effective for datasets exhibiting pronounced seasonal patterns. SARIMA offers enhanced flexibility in modeling complex time series by incorporating supplementary seasonal components. For example, when analyzing sensor data from an industrial machine that demonstrates consistent seasonal patterns—such as heightened vibration levels during specific operational cycles—a SARIMA model would be preferable than ARIMA for detecting anomalies. SARIMA successfully models and forecasts anticipated seasonal fluctuations, facilitating the identification of aberrations that genuinely signify aberrant or malfunctioning situations, as opposed to regular cyclical patterns. There is no seasonality in our industrial dataset, as it is stationary due to the data being insufficiently large and having been recorded over a short period. Some machines have 3 months of recorded data, and some have 6 months of recorded data. We attempted to examine seasonality in it, but it did not reveal any seasonality or trends. We then shifted our study to clustering techniques, which are described in the following sections. We examine the seasonality of MachineB, as shown in Figure 3.6.

As seen from the above explanation, our industrial dataset lacked predefined labels. This is the case with most industrial datasets; that is the reason AD in industrial processes is crucial. Additionally, before applying any ML model, it is a good approach first to understand the data both numerically and graphically. So all the DS prerequisite steps have been completed before applying the ML Model. If we briefly summarize the earlier analysis, it can be concluded that our industrial dataset lacked ground truth information, allowing us to use unsupervised ML techniques/algorithms to identify anomalies in machine operations.

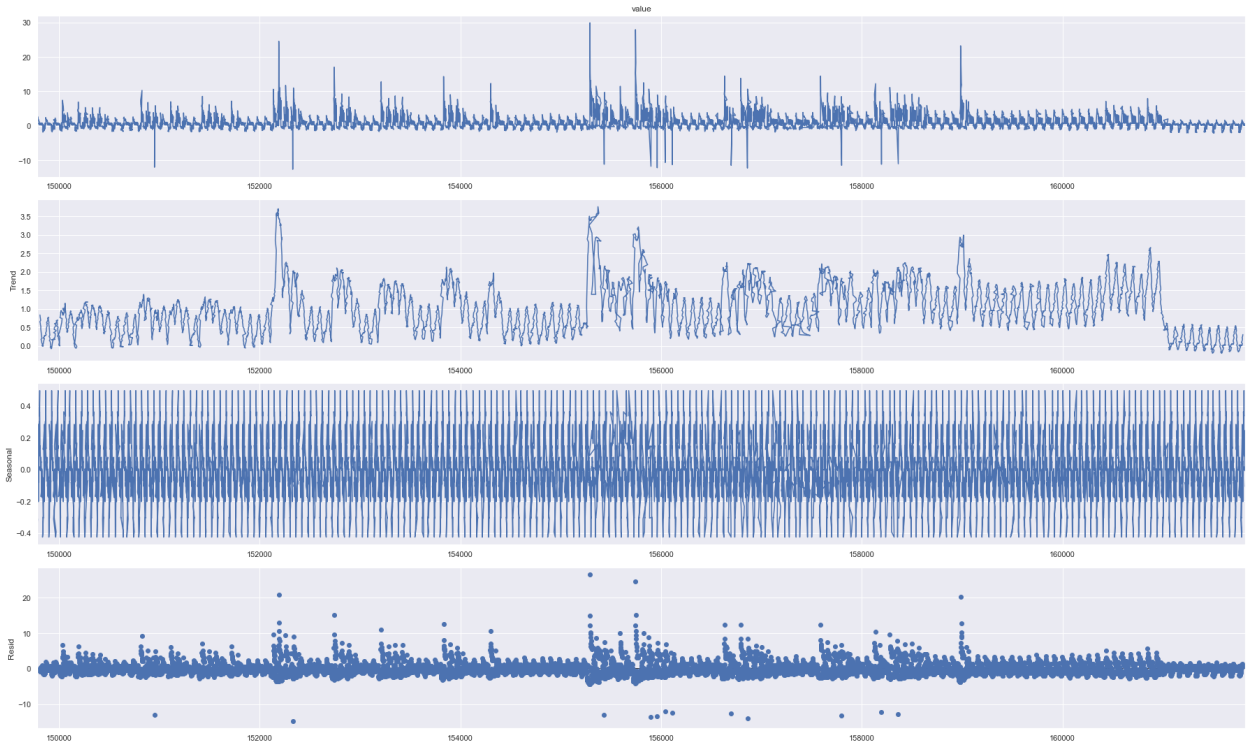

Figure 3.6: Seasonality analysis of MachineB

3.2 Marposs Anomaly Detection Use Case

As we discussed in Section 3, we filtered the dataset using different *indicator*. When we filtered the dataset using the FP: Ripple indicator, we found a completely different trend that is not available in any other machines, regardless of the time setting. The filtered trend schema is displayed in Figure 3.2 and shown graphically in Figure 3.7.

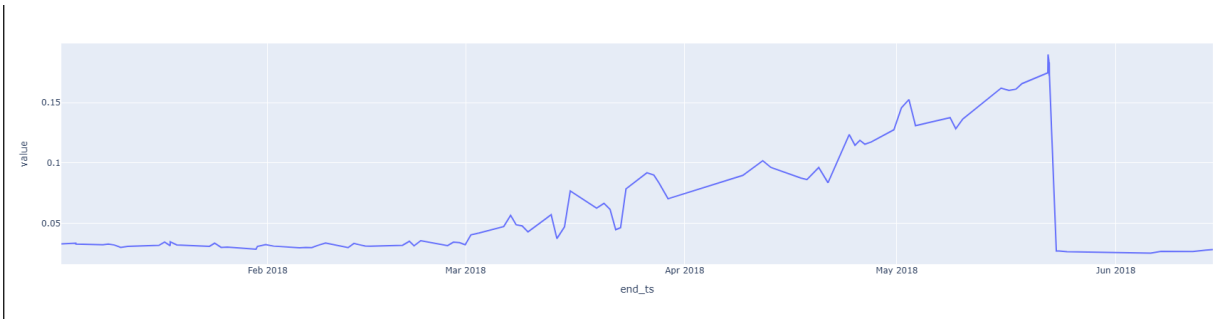

Figure 3.7: Marposs' anomalous use case trend

This anomalous trend began in February 2018 and continued until June 2018. The timestamp value is on the x-axis. We have measurement values on the y-axis in the *value* column of a dataset. We selected this as our baseline and anomalous trend, and then applied SOTA unsupervised ML algorithms.

3.2.1 Industrial Dataset Grouping

After preprocessing and statistical analysis of the industrial machines dataset, we further manipulated the industrial anomalous trend dataset. As *start_ts*, *end_ts*,

and *value* are the critical columns in the dataset, we manipulated these attributes to adjust the threshold of anomalous points—the procedure that creates different groups in a dataset and assigns a threshold to each group. Based on the threshold value, we can determine which group is anomalous or normal. The complete workflow of this grouping mechanism is:

- Initially, analyze the 10 machine cycles of MachineB by calculating the time differences between the *start* and *end* timestamps of the data, while distinguishing each machine cycle together with all its indicators (Ripple, Deviation, Area, Min, Max) simultaneously. Throughout each cycle, the values of all indicators (Ripple, Deviation, Area, Min, Max) for a single machine cycle are aggregated.
- Categorize the data based on the time intervals of the 10 cycles and aggregate the values for each machine cycle accordingly. The final data schema after this stage is shown in Figure 3.8.
- Calculate the mean of the value's interval_sum column. It will serve as the benchmark for identifying irregularities in each cycle. The hypothesis states that any combined value of a machine cycle that exceeds or falls short of this threshold will constitute an abnormality. This threshold might be assessed across a single cycle or multiple cycles.
- Plot the simple graph of the dataset.
- The plot with anomalies is now generated using the threshold criteria for each machine cycle.

	group_id	value's_interval_sum	contiguous_interval_count
0	1	0.033054	1
1	2	0.033657	1
2	3	0.032887	1
3	4	0.032290	1
4	5	0.032808	1
...	...	...	...
100	101	0.025352	1
101	102	0.026758	1
102	103	0.026721	1
103	104	0.027903	1
104	105	0.028309	1

105 rows × 3 columns

Figure 3.8: Grouped dataset schema

This procedure can also be implemented on other machines. The outcome can indicate whether the dataset contains more groups than some of those groups are

anomalous, based on the threshold for the aggregated value. The implementation of this grouping mechanism generates anomalies in the trend and identifies the specific points where anomalies can occur. This is evident in Figure 3.9. Algorithm 1 describes our custom grouping procedure, which determines the threshold for detecting anomalies in industrial processes.

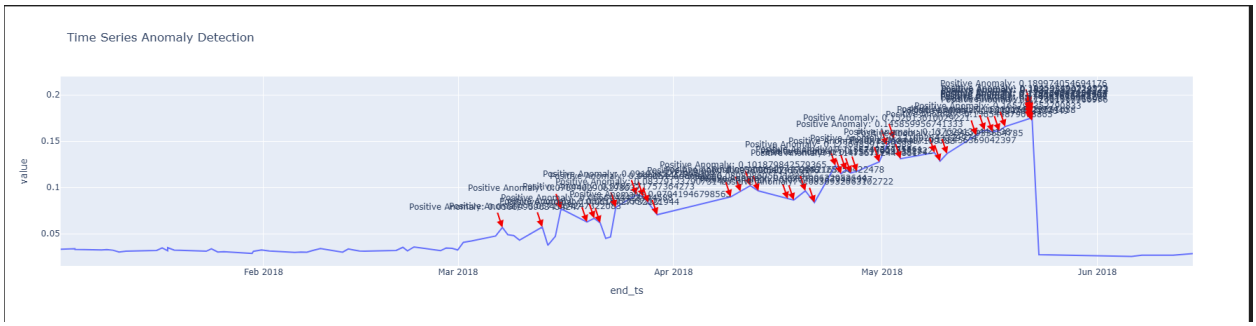

Figure 3.9: Anomalies identified using the custom grouping method

This grouping method also detects anomalies that are above and below the threshold range. We already discussed that this method is applied to different machines. Just for convention, we mark anomalies as red arrows that are below the threshold range and blue arrows that are above the threshold range. Its presentation is in Figure 3.10.

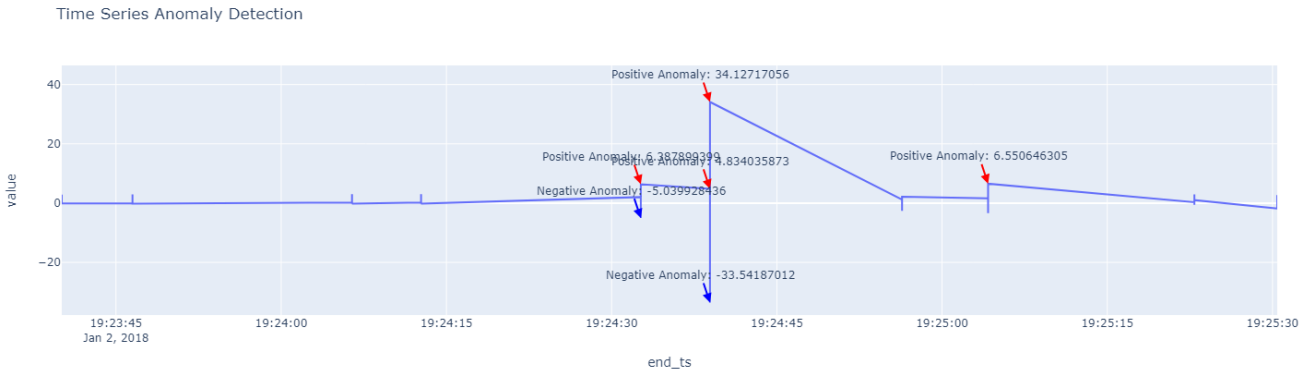

Figure 3.10: Anomalies identified using the custom grouping method on an industrial DT machine

Algorithm 1: Industrial Dataset Grouping

```
Initialize threshold ← 0;
Select the first ten machine cycles;
Extract data for the specified number of cycles;
Identify indicators: Ripple, Deviation, Area, Min, Max;
Sum up the values of all indicators with their fixed axes, tools, and processes;
Group the data by taking the time difference of each entry in a machine cycle
  and aggregating the values;
Calculate the RMSE value of the aggregated machine cycle's values;
Calculate the average RMSE values of each cycle to set the anomaly
  threshold;
if any value exceeds or falls below the averaged RMSE then
    Mark the cycle as anomalous;
else
    Repeat steps 1–9 for 20, 30, and other machine cycles;
    Plot the graph of the dataset with anomalies;
```

3.3 Clustering Techniques

Clustering algorithms are crucial for AD, as they group similar data points, allowing anomalies to emerge as outliers that deviate from typical patterns. These methods help identify data points that differ significantly from the norm, often indicating errors, rare events, or potential threats. The significance of clustering in AD can be summarized as follows:

- **Establishing Normal Behavior:** Algorithms like k-means, DBSCAN, and hierarchical clustering autonomously identify groups of similar data points, effectively modeling normal patterns in the dataset.
- **Identifying Deviations:** Once clusters are defined, points that fall far from any cluster can be flagged as anomalies due to their divergence from expected groupings.
- **Dimensionality Reduction:** Clustering simplifies complex datasets by aggregating similar instances, making it easier to isolate and analyze outliers in the residual space.
- **Augmented Feature Extraction:** By revealing latent structures and relationships in data, clustering enhances the effectiveness of downstream anomaly detection techniques.
- **Threshold-based Detection:** Anomalies can be detected by setting thresholds based on distance to cluster centroids (as in k-means) or cluster density (as in DBSCAN). Points exceeding these thresholds are treated as outliers.

Common clustering techniques used in AD include:

- **K-means:** Assigns data points to k clusters based on their distance from centroids; outliers typically lie far from any centroid.

- **DBSCAN:** Detects clusters based on data density, classifying sparse-region points as anomalies.
- **Hierarchical Clustering:** Builds a cluster tree (dendrogram) that supports multi-level anomaly detection with varying granularity.
- **Gaussian Mixture Models (GMM):** Assume data originates from a mixture of Gaussian distributions; data points with low likelihood under the learned distributions are considered anomalies.

After analyzing the industrial dataset, we determined that we can also utilize clustering techniques to identify anomalies in the dataset patterns. Previously, we implemented a custom grouping method to identify anomalies in both anomalous cases and normal industrial use cases. Additionally, three clustering algorithms —K-means, DBSCAN, and Isolation Forest (IF)—were applied to both normal and anomalous use cases to verify their applicability in industrial scenarios. K-Means and DBSCAN algorithms are implemented using the $scikit-learn$ package. $Scikit-learn$ [110] is a free and open-source ML framework written in Python that makes it easy and quick to mine and analyze data. It is built on top of foundational libraries, including NumPy, SciPy, and Matplotlib. It is used in both academia and industry to construct a wide range of ML algorithms and workflows.

3.3.1 K-Means Clustering

K-means clustering [111] is a widely used unsupervised ML approach that divides a dataset into k separate and non-overlapping clusters. The technique operates by repeatedly allocating data points to the closest cluster centroid and then updating the centroids by computing the average points assigned to each cluster. 'K' in K-means represents the number of clusters. The Elbow method [112] was used in advance to determine the optimal number of clusters for the dataset. It must be applied before applying K—means to avoid overfitting the ML model. In our dataset, the number of clusters is taken as three. The total variance among the observations in every cluster is known as the Within Cluster Sum of Squares (WCSS). It computes the squared difference between the two observations by measuring the separation between each observation and the centroid. The Elbow method representation is shown in Figure 3.11.

In Figure 3.11, we can see that the curve stops at point 3 and becomes smooth thereafter. Therefore, the maximum and final point of variation is found at point 3, indicating that the number of clusters will be three. We applied the k-means clustering algorithm after extracting the required number of clusters. The number of clusters could be changed depending on the specific industrial dataset. We first deploy k-means on the industry's anomalous case, as shown in Figure 3.7, and the implementation result is demonstrated in Figure 3.12.

The clusters are generated based on the values in the *value* column of the machine cycle. K-means was also implemented on other machines in the industry to see the variation in the clusters. Figure 3.13 illustrates the clustering results of different machines, including their complete manufacturing cycles and corresponding time intervals. These graphs aid in making decisions about anomalies. In Figure 3.12, the end_ts of a specific machine cycle is plotted on the x-axis, while its values in

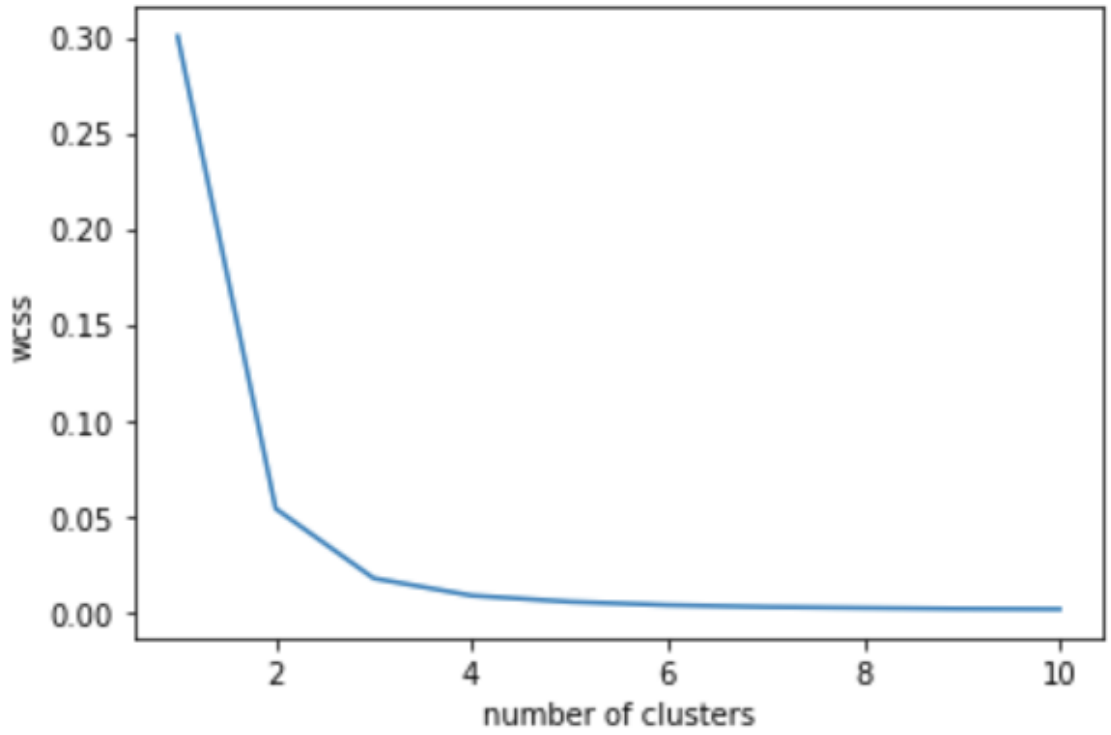

Figure 3.11: Elbow method for cluster identification

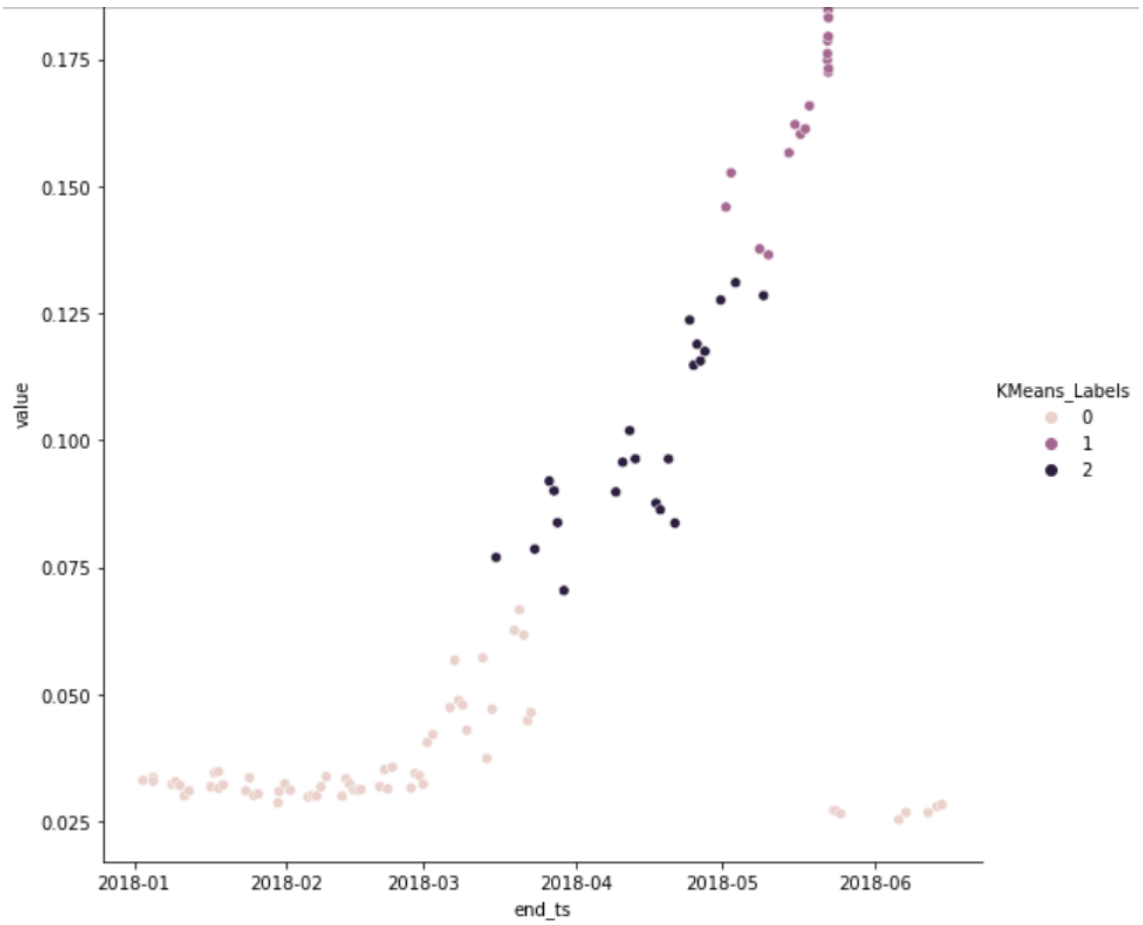

Figure 3.12: k-means on Marposs anomalous use case

the *value* column are displayed on the y-axis. The time is projected into different durations, allowing us to observe the anomalous effect over time. In Figure 3.12, the labels 1 and 2, in black and purple, are anomalous, confirming the industry's anomalous use case.

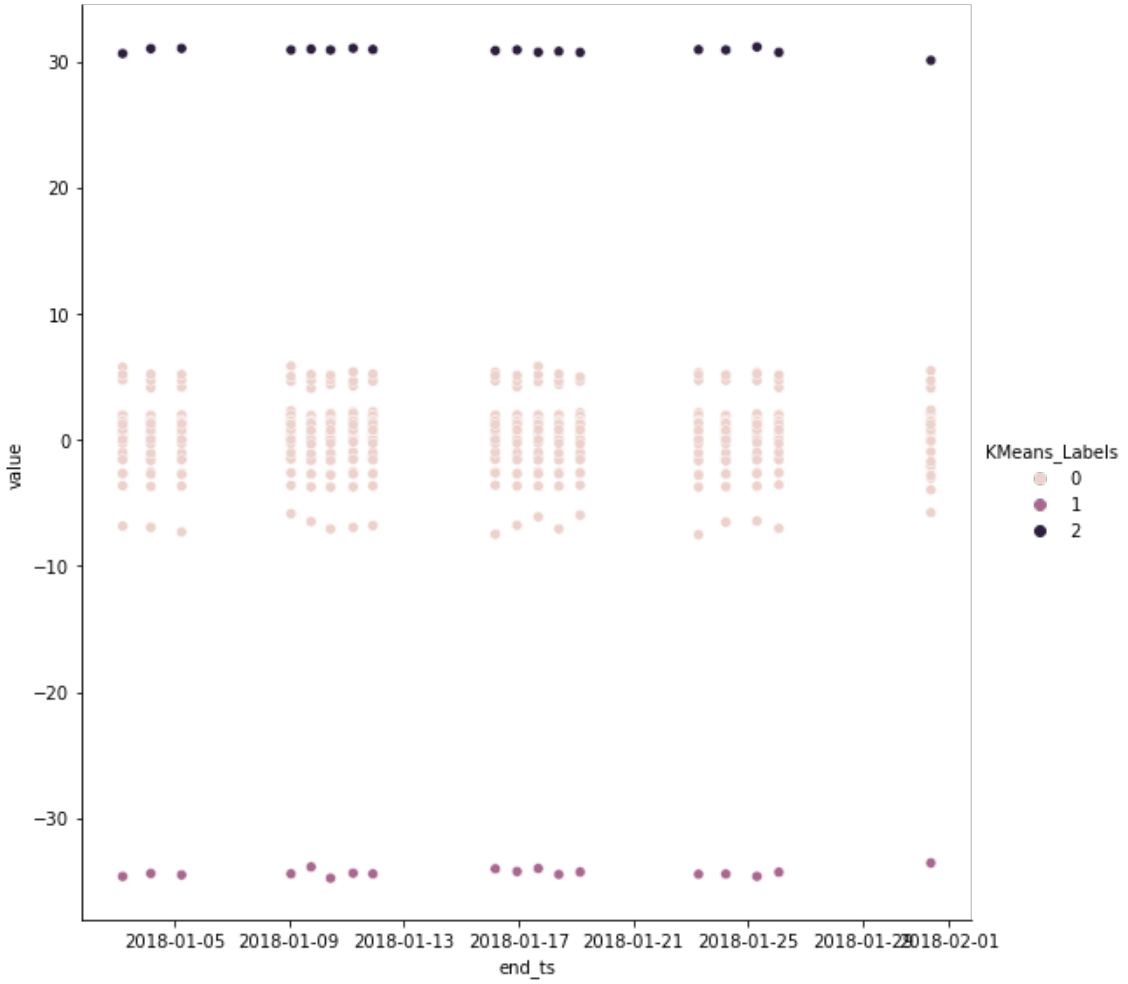

Figure 3.13: k-means on Marposs MachineA

A challenging task in cluster analysis is assigning labels to each point within the clusters, as these algorithms cannot do it automatically. A custom Python script was generated using the labels and centroids information generated by K-Means, which was employed with scatter plots to calculate and save abnormal indices, and label them on the plotted trend. The script can be found in my GitHub repository or in the Appendix section. Figure 3.14 displays the anomalous indexes, the number of anomalies, and their corresponding values for the industrial anomalous case. Our custom Python script enables us to record and annotate the anomalous indices in the dataset trends. It also recorded the values at which the anomalies occurred.

3.3.2 DBSCAN Clustering

Similarly, we have applied DBSCAN [113] to the industry's dataset. It is a widely used clustering technique in ML and data mining. Unlike the k-means algorithm, which necessitates pre-specifying the number of clusters, DBSCAN does not require

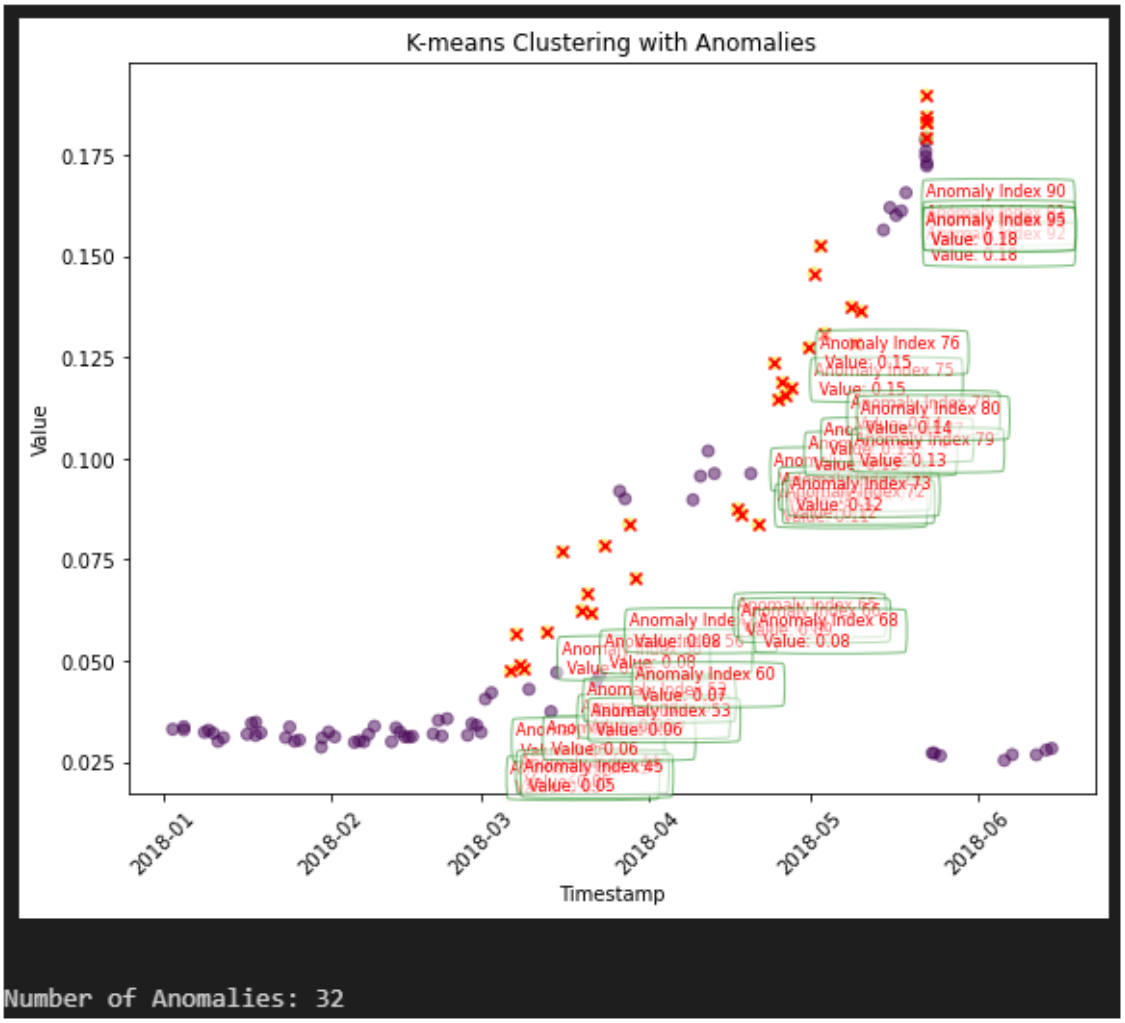

Figure 3.14: Anomalous labels indexing

specifying the number of clusters in advance. *eps* and 'min samples' are the only adjustable parameters in this algorithm. To fine-tune the sensitivity of the DBSCAN algorithm to anomalies and the level of clustering granularity, modify the parameters *eps* and *min_samples*. These parameters could be adjusted in different industrial use cases. So, the industries could adopt it at their ease. Noisy points are labelled as -1, while the generated labels are between 0 and 1. Reducing the amount of *eps* can result in more clusters and potentially enhance the detection of more minor abnormalities while also increasing sensitivity to outliers. Nevertheless, if the value of *eps* is excessively tiny, it cannot detect significant anomalies or lead to overly fragmented clusters. Increasing the value of *eps* can result in fewer clusters and the potential for merging outliers into larger, more significant clusters. It could potentially increase the difficulty of differentiating anomalies from regular data, mainly if they are relatively few or far away from the primary cluster. Similarly, increasing the number of *min_samples* necessitates that clusters must have a higher density to be identified. Applying this technique can effectively eliminate unwanted interference and minor variations in the data, enhancing its resilience against extreme values. Reducing the *minsamples* parameter can lead to more data points being classified as noise or outliers. While this approach may enhance the ability to detect abnormalities, it also carries the risk of overfitting if the dataset contains significant amounts of noise. The clustering graphs generated by this algorithm are the same as those generated by K-means. However, to evaluate and compare it with K-means, its performance index values are different, which will be discussed later in the performance evaluation section.

3.3.3 Isolation Forest Algorithm

Another significant method implemented on the industrial dataset is IF [114]. The IF algorithm processes randomly sub-sampled data in a tree structure using randomly selected features. Deeper samples within the tree are less likely to be anomalies since they necessitate more cuts to separate them. To segregate observations, the IF randomly selects a feature and then selects a split value within the range of the feature's maximum and minimum values. It also provides the anomaly score for each sample. This algorithm is an ensemble method that combines binary and binary search tree (BST) techniques. Optimising the parameters of this algorithm is crucial to achieving precise outcomes. The *contamination* parameter plays an essential role in managing this algorithm. This parameter quantifies the number of abnormalities present in any industrial data. Additionally, it can be advantageous to determine the threshold for the sample scores [114]. The functions $decisionfunction()$ and $predict()$ are utilised for additional significant purposes. The initial function will compute the anomaly score based on the indicator values at each timestamp. In contrast, the subsequent function will provide the label corresponding to that anomalous score. '-1' indicates an anomaly, while '+1' indicates a typical score. The identification of anomalies in IF on the industry's anomalous case is shown in Figure 3.15.

We implemented the IF algorithm on another public dataset, Numenta Anomaly Benchmark (NAB) [115], to compare the AD results. It is an innovative benchmark designed to assess the performance of ML algorithms used to detect anomalies in streaming and online applications. The NAB dataset consists of more than 50 labelled real-world and artificial time series data files, along with a unique scoring

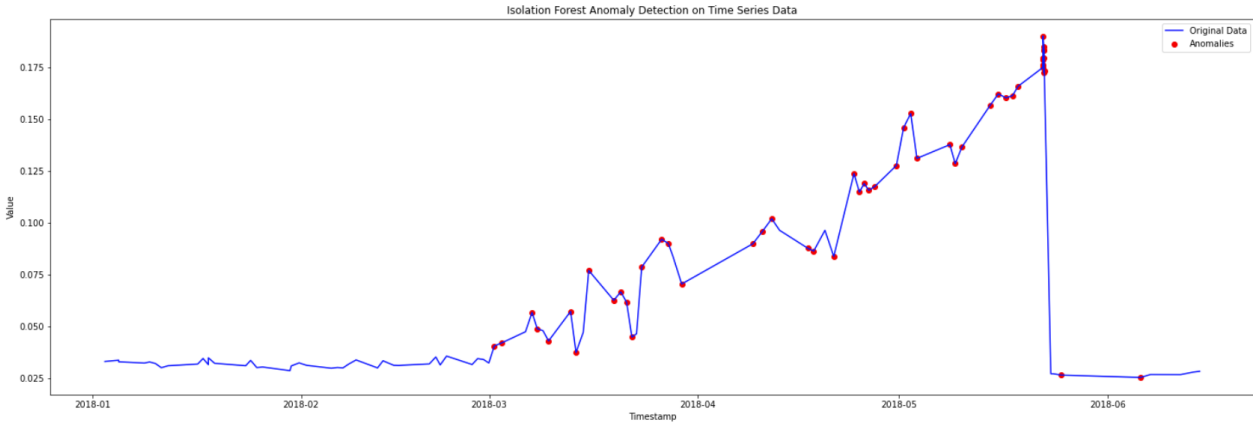

Figure 3.15: IF anomalies detection on Marposs anomalous use case

mechanism tailored explicitly for real-time applications. IF identified the anomalies where they occur in the NAB dataset. Figure 3.16 shows the implementation result of IF on the NAB dataset.

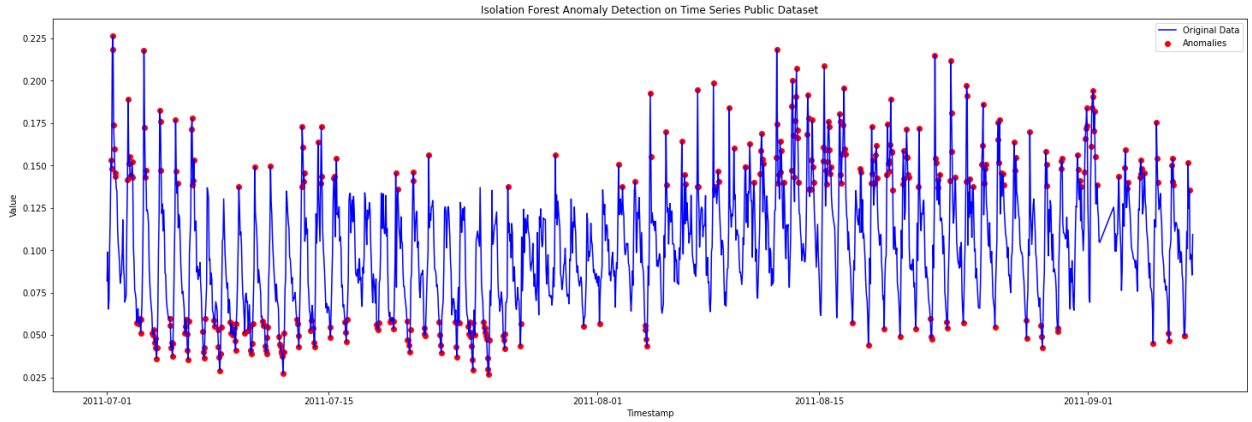

Figure 3.16: IF results on NAB dataset

3.3.4 Performance Evaluation of Clustering Algorithms

The IF algorithm effectively highlights anomalies, as K-means and DBSCAN do. From their results, we can say that these algorithms successfully predict anomalous points in the industrial dataset. These are unsupervised algorithms, so we cannot extract performance matrices such as F1 score, Recall, and Precision, etc. We evaluated the performance of these algorithms using three performance indices that are: Silhouette Score, Calinski Harbz Score, and Davis-Bouldin Score [116].

- **Silhouette Score**: The mean nearest-cluster distance (b) and mean intra-cluster distance (a) for every sample are used to compute the silhouette coefficient. The silhouette coefficient is $(ba)/max(a,b)$ for a given sample. Here, b is the separation between a sample and the closest cluster it does not belong to. One is the ideal value, while -1 is the worst. Clusters that overlap are shown with values close to 0. Since a different cluster is more comparable, negative values typically mean that a sample has been assigned to the incorrect cluster. In the anomaly case of industry, its value is 0.75.

- **Calinski-Harabasz**: The term "Variance Ratio Criterion" is another name for it. The score is calculated by dividing the sum of between-cluster dispersion by the amount of within-cluster dispersion. A higher Calinski-Harabasz score corresponds to a model with more distinct clusters. In our case, its value is 832.66.

- **Davis-Bouldin Score**: The score is determined by calculating the average similarity measure between each cluster and its most similar cluster. Similarity is the ratio of distances inside a cluster to distances between clusters. Therefore, clusters that are further distant from each other and have less dispersion will yield a higher score. The minimum score is zero, and lower values signify superior grouping. In our case, it gives a value of 0.405.

All these scores were also extracted for other machines in our industrial use case. Their performance index values justify the reliability of our implemented algorithms. In brief, these unsupervised ML models are reliable for detecting anomalies in industrial manufacturing processes. Advanced historical and exploratory data analysis helps identify the patterns and behaviors of expected anomalies. Custom grouping method works well before applying ML techniques to the industry's dataset. Thus, after evaluating the effectiveness of SOTA unsupervised algorithms on the industrial dataset, we enhanced their applicability and adaptability across the entire IEC continuum. The proposed DT reduces the manual implementation of ML algorithms in industrial processes. It's a comprehensive IEC-based solution that leverages the best practices of AI and MLOps. The proposed solution automated the AD process end-to-end, from data processing to model training, and optimized the deployment of the trained ML model on edge devices. The next chapter will discuss in detail the architecture and implementation of the proposed DT framework.

Chapter 4

Design and Implementation of the proposed Framework

4.1 C2E-AIOps Framework Description

From now on, we will address all the remaining research questions highlighted in Section 1.1.

DT technology meets many of Industry 4.0's primary needs by providing real-time, data-driven simulations of physical assets, processes, and systems. DT is crucial in Industry 4.0, which focuses on networked systems, cyber-physical integration, intelligent automation, and data-driven decision-making. First, DT enhances real-time monitoring and predictive maintenance, which helps industries reduce downtime and extend the lifespan of their assets. DTs replicate current circumstances and forecast future failures by constantly synchronizing with sensor data from machines and manufacturing lines. This meets the demand for proactive and self-sufficient maintenance plans. Our proposed DT framework highlights real-time data acquisition and data analytics using the latest cloud technologies, including Azure ML, AutoML, and IoTCentral, for data manipulation.

Second, DT helps customize and optimize processes. In a constantly evolving manufacturing environment, DT enables you to test and refine production processes through simulations before implementation, saving time and money. This adaptability aligns with the Industry 4.0 goal of creating mass customization while maintaining efficiency. Third, DT facilitates data integration and analytics by serving as a single platform for consolidating data from IoT devices, ERP systems, and other sources. This comprehensive view aligns with Industry 4.0's need for data-driven insights to inform better business decisions. Lastly, DT enables people and machines to collaborate and for machines to interact with one another. Workers can use virtual models to teach, fix problems, or make strategic decisions without stopping real production. To make industrial systems more innovative, safer, and more flexible, digital enhancement, as highlighted above, is necessary. Our proposed DT framework is adaptable and capable of performing the operations mentioned above. The proposed DT framework is illustrated in Figure 4.1.

Some latest cloud-based technologies used in the proposed DT are described below:

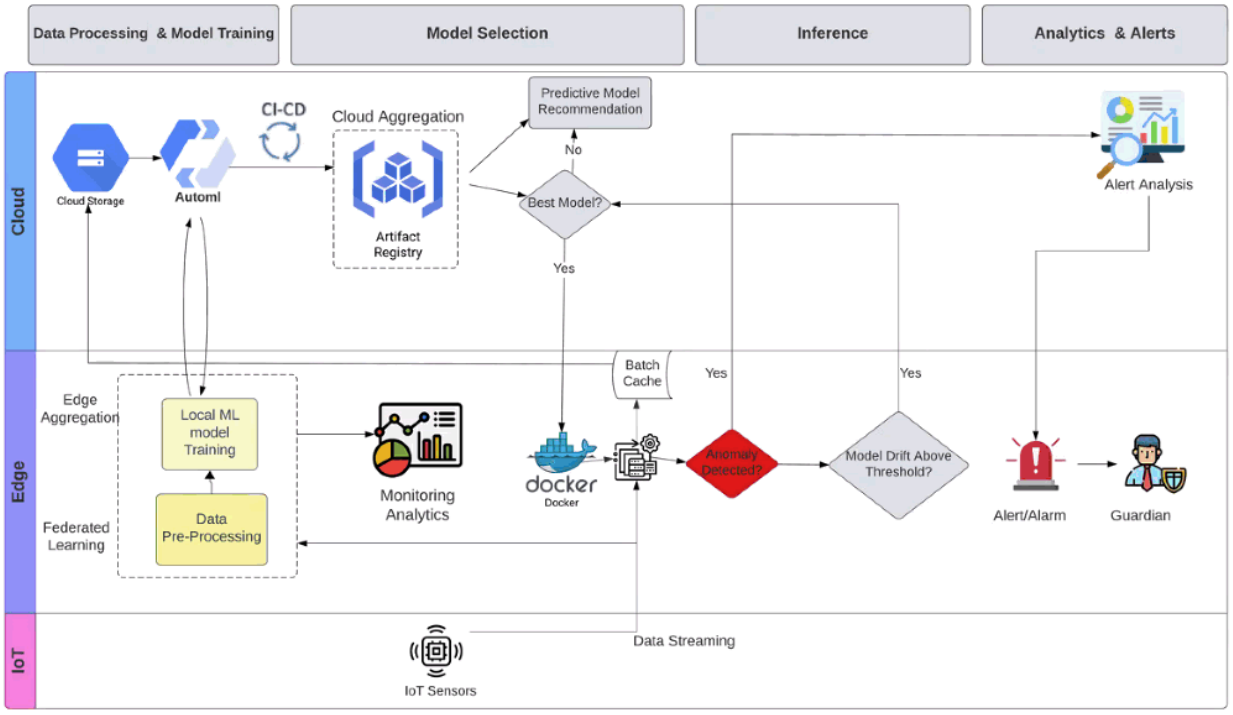

Figure 4.1: Proposed AI and MLOps powered DT framework for Marposs S.p.A

4.1.1 Components of C2E-AIOps Framework-Cloud Layer

- **Microsoft Azure** [117]: It offers an extensive framework for creating and managing AI and MLOps applications. It provides services such as Azure ML, Cognitive Services, and OpenAI on Azure to facilitate custom model building and pre-trained AI functionalities, including vision, speech, and language comprehension. Developers may utilize prominent frameworks such as TensorFlow, PyTorch, and Scikit-learn for large-scale model training. Azure ML streamlines the complete machine learning process, encompassing data preparation, Automated Machine Learning (AutoML), pipeline development, and model deployment. It additionally facilitates version control, experiment monitoring, and reproducibility. Models can be distributed to endpoints or containerized using Azure Kubernetes Service (AKS). Azure seamlessly integrates with CI/CD tools such as Azure DevOps and GitHub Actions, facilitating efficient MLOps workflows. The platform guarantees scalability, security, and operational efficiency, rendering it suitable for enterprise-level AI applications.

- **Azure IoT Hub** [118]: It is also a Microsoft cloud-based service that facilitates bi-directional communication between IoT applications and devices. It supports millions of connections and enables secure data exchange via protocols such as MQTT, AMQP, and HTTPS. It includes features like device twins and cloud-to-device messaging for secure device management.

- **Azure ML/AI Studio** [119]: It is a web-based integrated development environment (IDE) offered by Microsoft as part of Azure ML. It serves as a user-friendly platform for constructing, training, deploying, and overseeing ML models. The AI studio supports both code-first methodologies (Python

and Jupyter notebooks) and no-code/low-code interfaces (via a drag-and-drop designer), making it accessible to data scientists, ML developers, and business analysts. It facilitates constant data exploration, experiment monitoring, pipeline development, and model assessment. Integrated tools for AutoML, responsible AI, and model monitoring enhance the efficiency of the MLOps lifecycle. Azure ML Studio facilitates collaboration, scalability, and governance, enabling teams to create AI solutions in a secure, enterprise-grade environment swiftly.

- **AutoML** [120]: It is an vital part of Azure ML/AI studio that lets users build and improve ML models automatically, without having to do a lot of manual coding or having a lot of experience with DS. It accelerates the process of selecting an algorithm, designing features, fine-tuning hyperparameters, and evaluating models. Users can provide AutoML with a dataset and a target column, and it will train and compare various models to identify the best one based on the specified metrics. Azure AutoML can be used for tasks such as classification, regression, and time-series forecasting. It works well with the Azure ML Studio interface for both code-based and no-code workflows. Because of this, it is ideal for accelerating model development in business AI projects while maintaining accuracy, transparency, and scalability.

- **Azure Blob Storage** [121]: It is Microsoft's cloud-based object storage service, made to hold large amounts of random data like text, photos, videos, backups, and logs. It provides flexible, long-lasting, and secure cloud storage for various purposes, including analyzing large amounts of data, running ML processes, streaming media, and archiving old files. Blob storage allows setting different data tiers (hot, cool, and archive) to achieve optimal cost-effectiveness based on usage. It also integrates with other Azure services, such as Azure Data Factory, Azure ML, and Azure Synapse Analytics, making it easy to add data, process it, and train models. Blobs include built-in tools —such as lifecycle management, encryption, and role-based access control (RBAC) —that enable developers to work with them via REST APIs, SDKs, or the Azure Portal. We have an industrial time-series dataset. We used a blob storage service integrated with the industry's edge and IoT devices to store the dataset for processing using the selected Cloud technologies.

- **Azure IoT Central** [122]: It is a fully managed low-code Platform-as-a-Service (PaaS), a platform developed by Microsoft, allowing customers to rapidly construct, deploy, and administer IoT applications without extensive cloud proficiency. It streamlines device connectivity, facilitates real-time monitoring, and enables remote management, while offering integrated dashboards, rule-based notifications, and secure data integration with additional Azure services. IoT Central is engineered for scalability and user-friendliness, facilitating the rapid development of IoT solutions. This makes it suitable for enterprises aiming to optimize operations and gain insights from connected devices with minimal infrastructure requirements.

- **GitHub** [123]: GitHub is a cloud-based platform for version control and collaborative software creation. It was built on Git, which is a popular distributed version control system. It enables developers to host, control, and track changes

to code repositories. This makes it easier for people from different teams and locations to collaborate on projects. GitHub offers features such as branching, pull requests, issue tracking, project boards, and actions for CI/CD, which enable DevOps processes to run smoothly. It is widely used in both open-source and business settings because it can automate tasks, facilitate continuous integration, and integrate with tools like Visual Studio Code, Docker, and Azure. GitHub encourages community development and is a key platform for sharing, reviewing, and releasing software in modern development environments. We implemented our DT framework's CI/CD pipeline using GitHub Actions.

- **GitHub Actions** [124]: It is a robust CI/CD automation solution integrated within GitHub. It enables developers to automate workflows for constructing, testing, and deploying code directly from their GitHub repositories. GitHub Actions employs YAML configuration files located in the .github/workflows/ directory of a repository. Each file outlines a workflow, comprising a collection of tasks. Each job operates within a virtual environment (such as Ubuntu, Windows, or macOS) and includes steps that execute shell commands or predefined activities (reusable tasks). Workflows are initiated by events such push, pull request, problem, or according to a schedule (cron). For instance, when code is committed to a branch, a workflow can autonomously execute tests, generate artifacts, or deploy to a server or cloud platform like as Azure or AWS. GitHub offers a marketplace for community-contributed actions, and developers have the capability to construct their own. Confidential information can be securely kept for utilization in workflows (e.g., API keys). GitHub Actions streamlines DevOps by integrating automation right into the repository.

In our proposed DT framework, the trained ML models are deployed on our industrial edge and IoT device. Let's provide an overview of the tools used to deploy the ML model at the edge.

4.1.2 Components of C2E-AIOps Framework-Edge Layer

- **Azure IoT Edge** [125]: It is a Microsoft service that provides cloud capabilities to IoT devices at the edge of the network, reducing latency and bandwidth. It supports Linux and Windows platforms and integrates securely with Azure services, making it ideal for industrial automation, remote monitoring, and smart environments.

- **Docker Containers** [126]: It is an open-source platform that simplifies application development, deployment, and execution through the use of containers. It supports rapid scaling, microservices architecture, and isolated testing environments. Docker images are widely compatible across operating systems and orchestration supports, making it a crucial tool for modern software engineering and system deployment.

- **IoT/Industrial PC**: This is a lightweight system, PC, or industrial in-house device running on Windows or Linux OS, where the packaged model, containerized through Docker, has been implemented. The Industrial PC, connected to industrial systems, utilizes real-time data inputs to infer ML models. It can be optimized using frameworks such as ONNX Runtime and Azure IoT Edge

modules and may include a local API for seamless integration. We used ONNX Runtime and RESTful API integration for edge-layer deployments.

- **Federated Learning** [127]: It is a decentralized ML methodology that enables model training across multiple devices or servers without exchanging raw data. Rather than transferring data to a central server, each participating node trains a local model using its own data and transmits only the model updates (e.g., gradients or weights) to a central aggregator. This methodology improves data privacy, security, and regulatory adherence, particularly in sensitive sectors such as healthcare, banking, and mobile technology. Federated learning minimizes data transfer burdens and facilitates tailored models while safeguarding user data on the originating device. It is often integrated with privacy-preserving technologies, such as differential privacy and safe aggregation, making it a crucial enabler of collaborative AI in remote and privacy-sensitive contexts.

4.2 Architectural Framework Description

This section will address the five research questions: RQ4-RQ8. It describes the architectural details of the proposed industrial DT. Many industrial prototypes are off-the-shelf systems as they did not make it up to the production system. Our proposed DT framework is implemented in a manufacturing industry, Marposs S.p.A. [44]. It provides a comprehensive digital overview of cyber-physical systems in industry. This framework is a complete IEC Continuum. It is composed of three layers: IoT, edge, and cloud, as illustrated in Figure 4.1. The data integration workflow started from bottom to top (IoT to Cloud), and framework deployment and implementation started from top to bottom (Cloud to IoT). Horizontally, we can see the layers view; vertically, the main procedures or operational blocks; and the technologies involved are highlighted. At the IoT layer, we have IoT devices. It can be Raspberry Pi kits, FPGA boards, mobile phones, tablets, etc. But in our case, we have a small industrial PC running Windows and the Linux subsystem.

4.2.1 End-to-end Data Processing Pipeline

The data workflow starts from the IoT layer. Our industrial PC continuously sends data or generates machine cycles after passing through the edge layer, and the data is stored in Blob Storage, particularly created for industrial PC's data. This IoT-based data has also been collected in the Azure IoT Central application. The industrial dataset is in JavaScript Object Notation (JSON) format. This dataset includes data collected by a Marposs measurement system that monitors various dimensional characteristics of a mechanical part produced by a manufacturing line (e.g., camshaft production). For each characteristic measured, it returns the following attributes: *deviceid*, *Serial_no*, *Instrument_name*, *Timestamp*, *Measurement_Attribute*, *Measurement_Value*. Each characteristic value, identified by a *Measurement_id* collected on production parts, is timestamped. The prediction is made on the target *Measurement_Value* column. The ML task used for this dataset is regression. The industrial PC continuously sends data to the industrial edge system, which then forwards it to Azure IoT Central. The data transfer schema is illustrated graphically in Figure 4.2. The *Data Export* feature in Azure IoT Central is used to

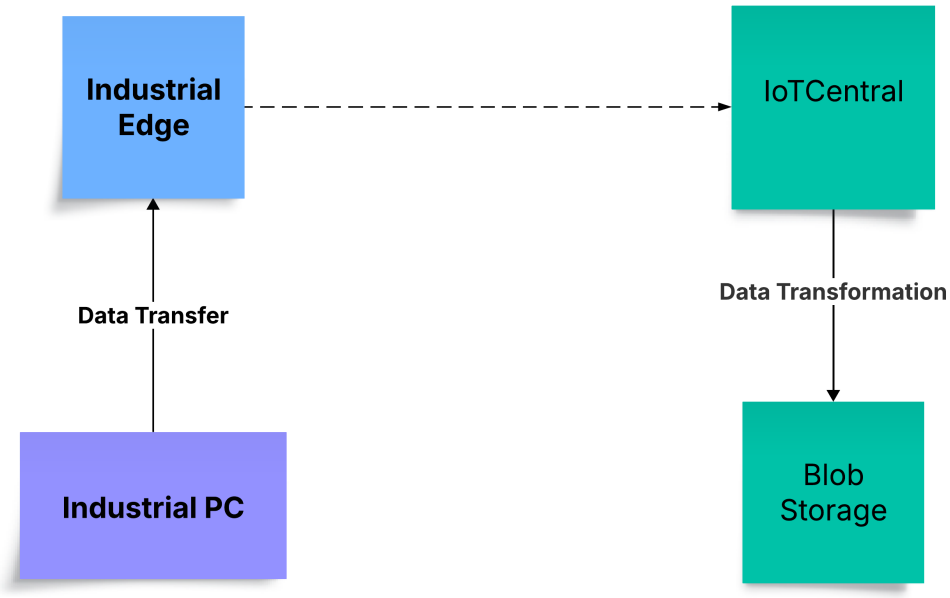

Figure 4.2: Data processing pipeline on MS Azure

transform and prepare the dataset for training the ML model. We used jQuery with the *iotc* package to convert the JSON data into a tabular format, since AutoML cannot process unstructured or non-tabular data. The transformed data is stored in datastores, a feature available in Azure ML Studio, and uploaded to AutoML for ML model training. All the necessary steps for creating an ML model, such as data preprocessing, normalization, feature extraction, dataset splitting, model training, testing, and evaluation, are performed by AutoML, resulting in the best ML model with higher accuracy.

4.2.2 Dataset Transformation

We have used an Azure IoT Central industrial application instance that is linked with the IoT data management of the product MAINDO. Our Blob storage is integrated with this IoT Central instance. In its *Data Export* feature, we did the data transformation of IoT data coming from IoT device as illustrated in Figure 4.2. The overview of this feature is in Figure 4.3.

When clicking on the *Edit* icon, we can see the data transformation query written in JQuery. This query aims to transform and structure incoming telemetry data from an IoT device for subsequent processing or visualization. This process consists of two primary stages: metadata extraction and measurement data mapping. Initially, the query examines the telemetry payload to identify specific fields of interest by comparing their names with established labels. The fields comprise a unique device identifier, a timestamp indicating the data-generating time, a serial number for traceability, and an instrument label specifying the source or context of the measurement. The values are obtained by identifying the element in the telemetry array with the specified name and extracting its associated value. After metadata extraction, the query proceeds to the second stage, where it processes an array of measurement values in the telemetry data. This system retrieves a set

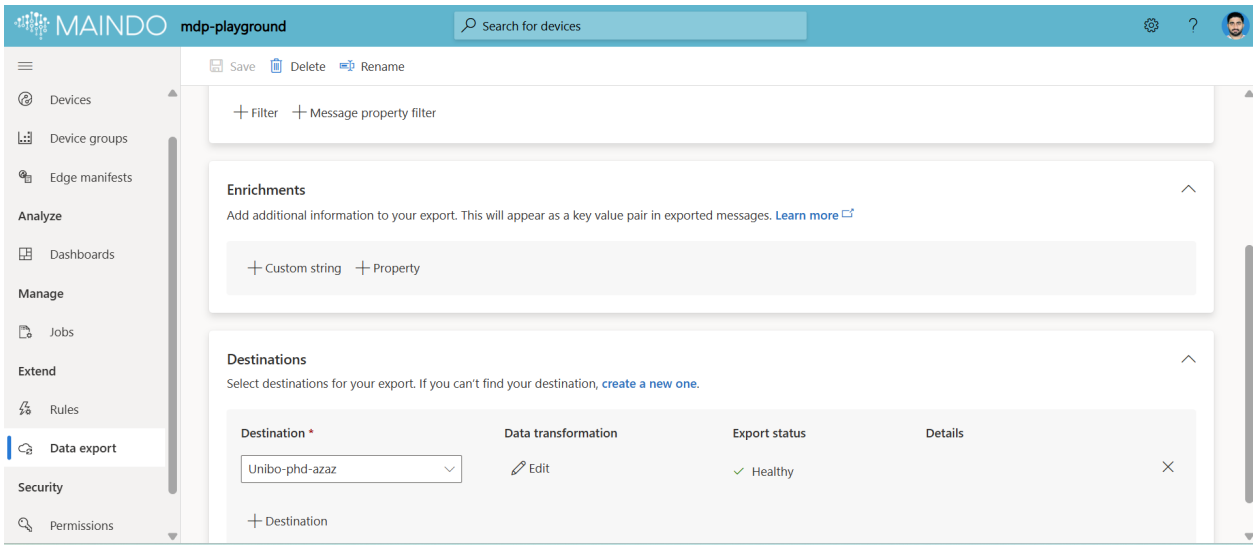

Figure 4.3: Data transformation at IoT Central's data export feature

of measurement entries, each potentially comprising diverse sample readings and descriptors. This collection's entries feature a query that extracts three essential components: a measurement identifier, the numerical reading of a sample value, and an attribute that characterizes the measurement. This step selectively extracts pertinent information from a potentially larger and more intricate data structure. The metadata and mapped measurement data are ultimately combined into a single, cohesive object. This yields a structured and semantically coherent representation of the telemetry, integrating critical context (including device identity and timing) with actual sensor readings and descriptive labels. We represent the telemetry payload as a set of name-value pairs

$$T = \{(n_i, v_i) \mid n_i \in N, v_i \in V\} \tag{4.1}$$

Where:

- n_i is the name of the telemetry field (e.g., "DeviceId")
- v_i is the corresponding value
- N is the set of possible field names
- V is the domain of values (e.g., strings, timestamps, floats)

The extraction of a specific field from telemetry data can be modeled as a filtering function:

$$f_{\text{extract}}(T, n_j) = v_j \quad \text{where } (n_j, v_j) \in T \text{ and } n_j = \text{"DeviceId"} \tag{4.2}$$

Mapping Measurement Values Let $M = \{m_1, m_2, \ldots, m_k\}$ denote a collection of measurement entries. Each measurement m_i contains structured nested values.

We define a transformation function g that extracts relevant measurement components:

$$g(m_i) = (m_i[\text{“K2001”}],\ m_i[\text{“Samples”}][1][\text{“K0001”}],\ m_i[\text{“Samples”}][1][\text{“K0002”}]) \tag{4.3}$$

The completely transformed measurement set is:

$$M' = \{g(m_i) \mid m_i \in M\} \tag{4.4}$$

Merging Metadata and Measurements The final structured output O is a union of metadata and extracted measurements:

$$O = f_{\text{meta}}(T) \cup g(m_i) \tag{4.5}$$

Where:

- $f_{\text{meta}}(T)$ extracts device metadata such as ID, timestamp, serial number, and instrument.
- $g(m_i)$ provides the transformed measurement data.
- $\cup$ denotes the union (or merge) operation on key-value maps assuming disjoint keys.

After this data transformation, when new machine cycles are generated from an Industrial IoT device, we receive the transformed tabular data, ready to feed into Azure ML/AI Studio for necessary ML model training. This completes the end-to-end data collection and transformation pipeline.

4.2.3 Data Registration on Azure ML/AI studio

First, we created a workspace on Azure ML/AI Studio with the name *trainmodels*. In Azure ML/AI studio, we take advantage of *Datastores* feature where we can resgitser our transformed data recorded in the Blob Stroage. The overview of *Datastores* feature illustrated in Figure 4.4. Our *Marposs* blob has been registered in the datastores. We also registered some other machine datasets in this feature. You can see other features of Azure ML/AI studio in Figure 4.4. *Automated ML* is one of the feature that we used for the dataset training. In *Data assets* tab, we uploaded the dataset, that will be used for the ML model training. *Data*, *Compute*, *Jobs*, *Environments*, *Pipelines*, *Models*, and *Notebooks* are all the components of Azure ML/AI studio that were used in our deployment.

4.3 AI/ML Models: Automated Training on Azure Cloud

We used the Automated ML feature to train ML models. Firstly, we registered our industrial dataset in *DataAsset* with the name *Marposstestdevice3*. It can be seen in Figure 4.5. There were no strict requirements for choosing the name; we could have chosen some other one as well.

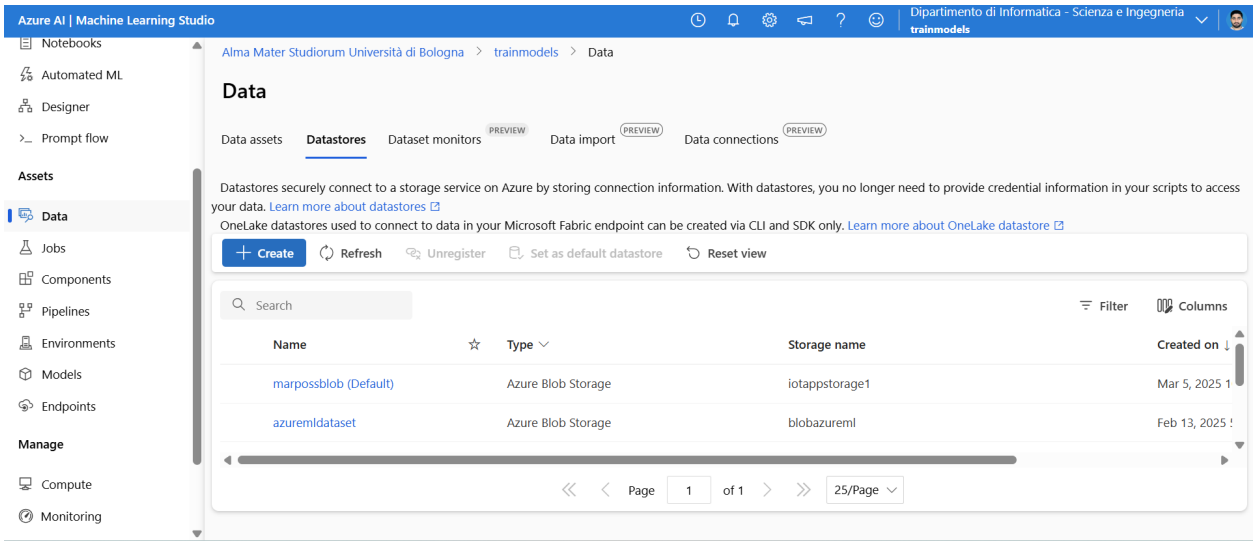

Figure 4.4: Datastores overview

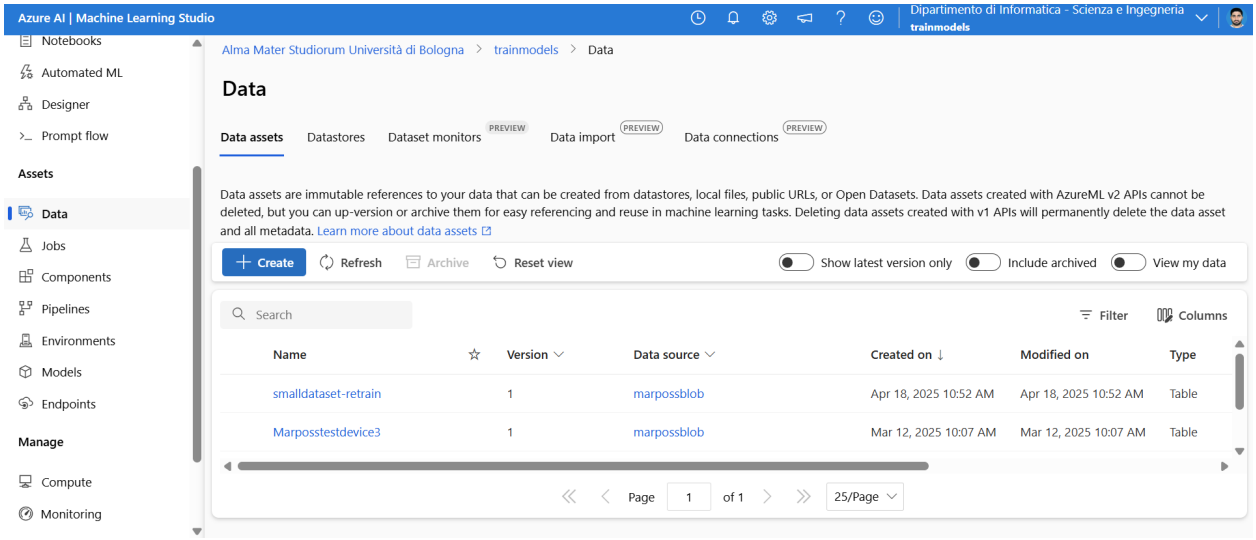

Figure 4.5: Dataset registration in data assets

4.3.1 Compute Instance Creation

Before creating a job, we first created a compute instance in Azure ML/AI Studio that executed all the jobs. In our case, the compute instance features are: $Standard\_E4ds\_v4$ $4cores, 32GBRAM, 150GBdisk$). The compute instance can be viewed in Figure 4.6.

4.3.2 AutoML Job Creation

Here, AutoML came into play, where we created a pipeline-based job by clicking the create button in the *Jobs* section. *Create-a-job* section can be seen in Figure 4.7. There are three options to create a job, but we chose the first one. It trained multiple ML models automatically by just giving some configuration parameters manually using Azure ML/AI Studio IDE. The parameters that can be set during this procedure are seen in Figure 4.8.

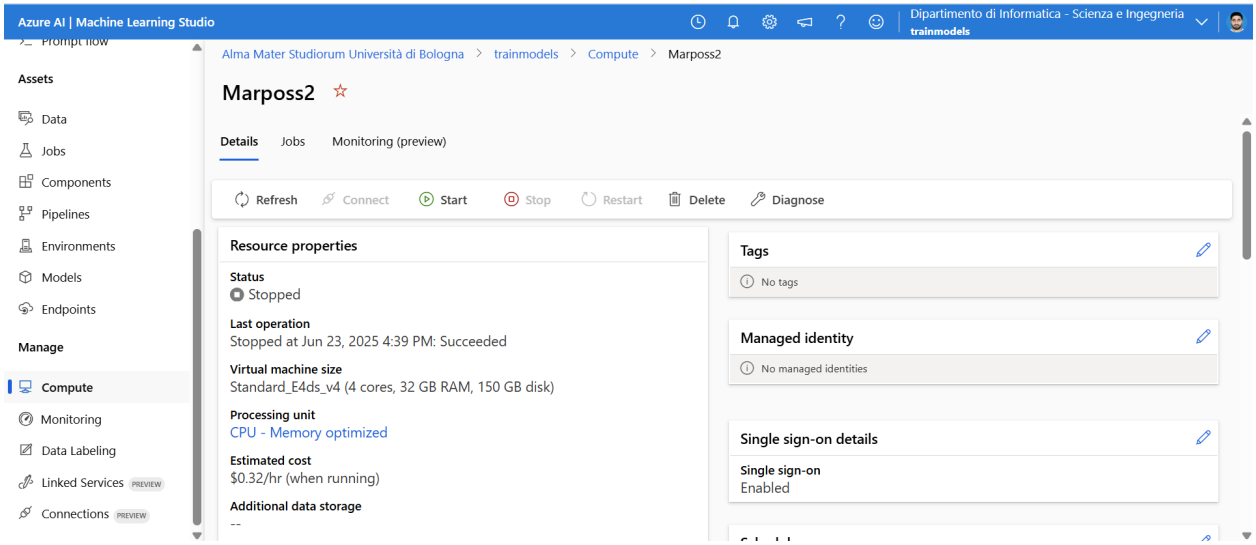

Figure 4.6: Compute instance creation

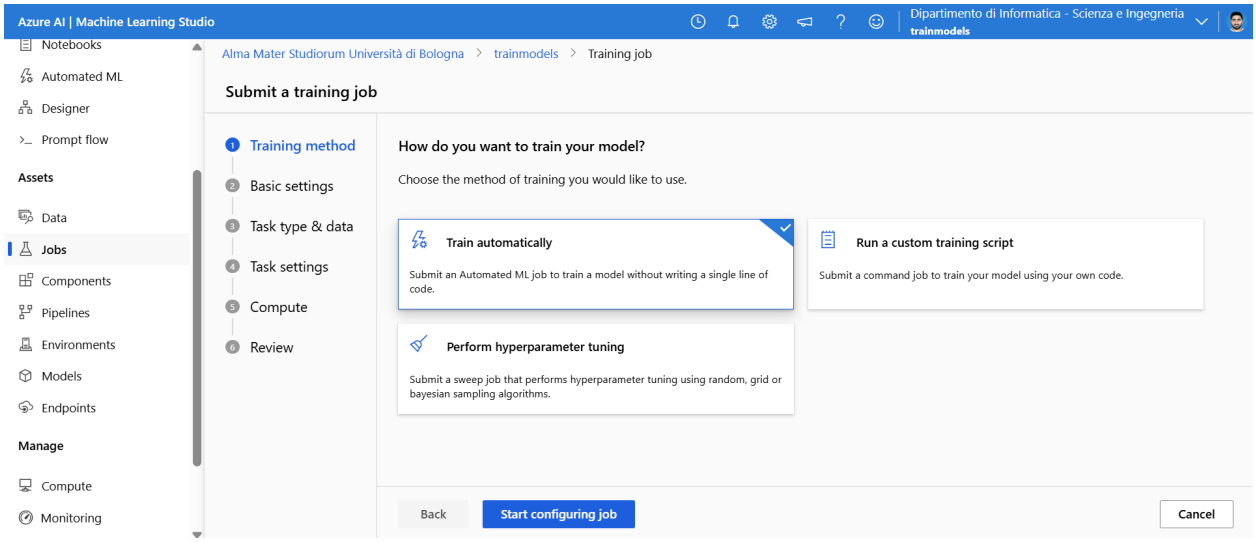

Figure 4.7: AutoML job creation

Firstly, the type of ML task must be selected. We specified the Regression task based on our industrial dataset. Secondly, the registered dataset, Marposstestdevice3 from *Data Assets* feature has been selected. The AutoML feature validated the data and selected the target column for prediction. In this phase, all ML steps, data splitting, data featurization, early stopping, and performance matrix selection, have been taken care of by this AutoML feature. At the end, the job has been submitted, and the final overview of the job is in Figure 4.9. This will start the job, and the average time to complete a job is 10-12 minutes.

4.3.3 Artifacts Generation

After the pipeline is completed successfully, the best ML model has been generated. The relevant performance matrix or indicator is extracted and can be registered in the *Models* section of Azure ML/AI Studio. The completed job is shown in Figure 4.10. The completed job name is *testazurefunction12*, as shown at the top of

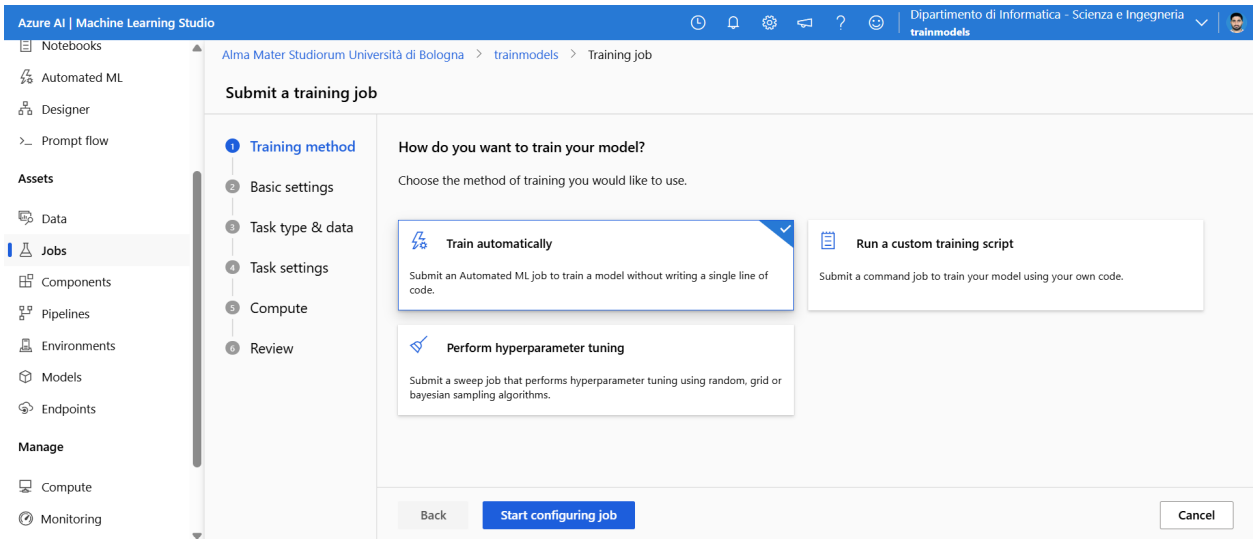

Figure 4.8: AutoML job configuration

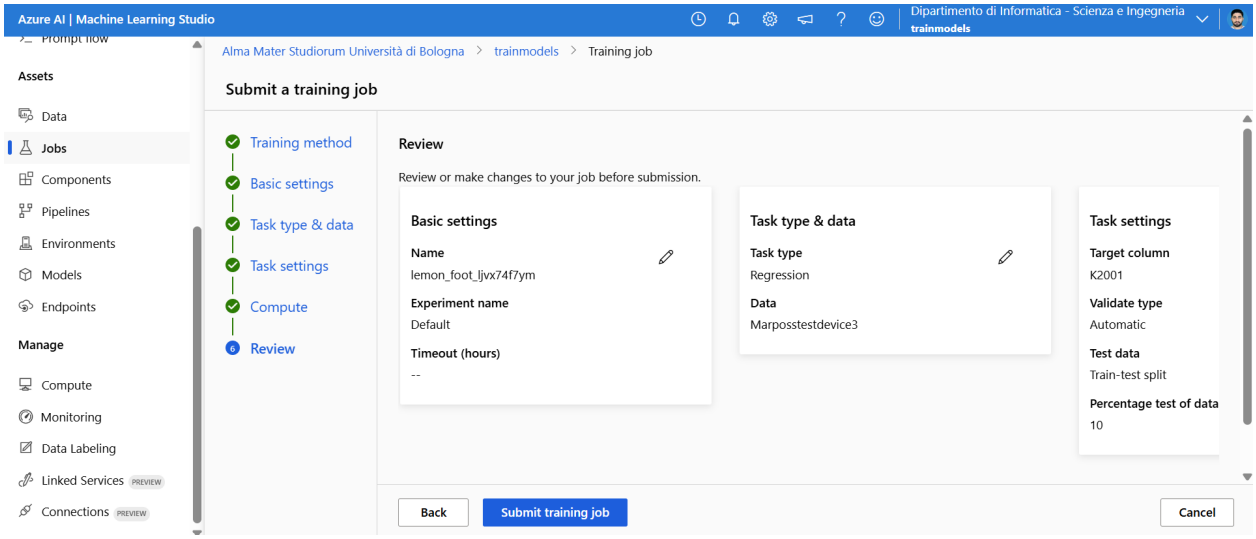

Figure 4.9: AutoML job submission

Figure 4.10. On the right side, the dataset used, the best model summary, and the model's R2 score are highlighted. On the right, the current job status and the total computation time of the job are shown. It provides a comprehensive view of an ML process through a no-code workflow. Internally, it also trained multiple ML models corresponding to each child's job. AutoML evaluates the accuracy and performance of all these child ML models, and ultimately gives us the best model. The child's jobs are shown in Figure 4.11.

The most crucial element in the ML model training is the generation of *.pkl* file, requirements file (requirements.txt), and environment file (.env). A *.pkl* file, or Pickle file, is a file format utilized in Python for the serialization and storage of Python objects on disk. Serialization in Python is handled by the built-in *pickle* module, which enables the storage of objects such as dictionaries, lists, models, and custom classes in a binary format for future use. *.pkl* files are primarily utilized for maintaining the state of objects, enabling their rapid reloading without the necessity of recreating them from the beginning. In ML workflows, trained models

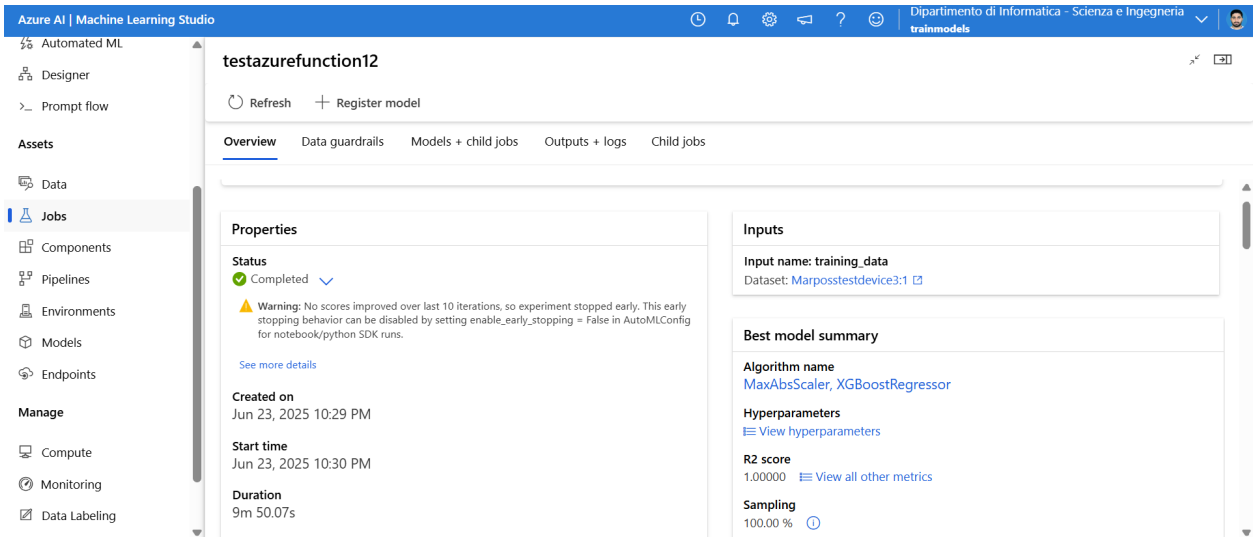

Figure 4.10: AutoML job completion

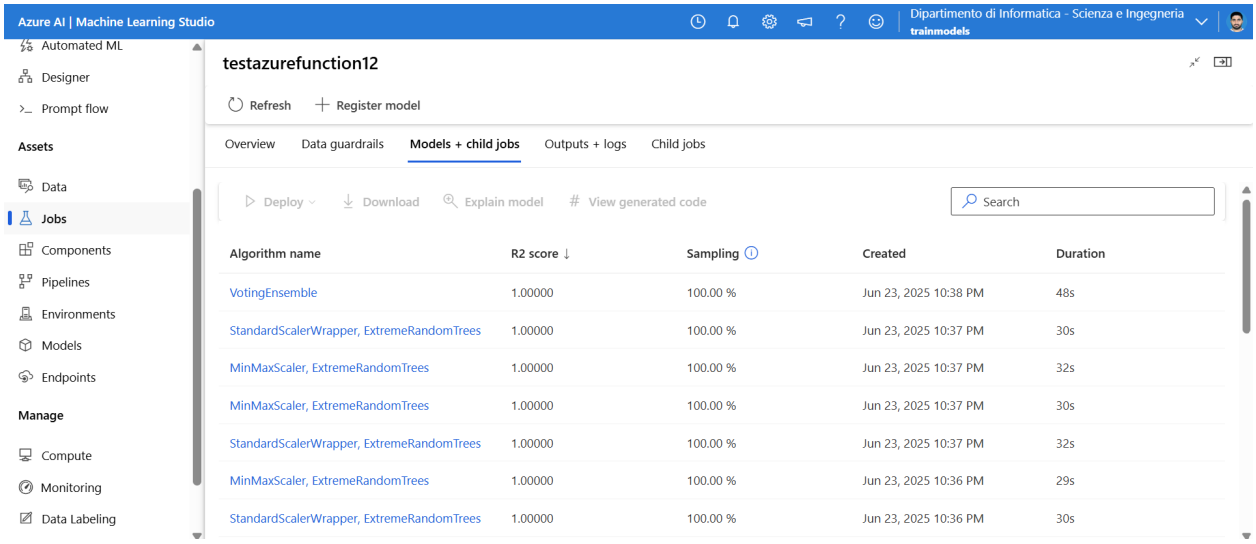

Figure 4.11: AutoML child jobs overview

are commonly saved as *.pkl* files to prevent the need for retraining and to enhance efficiency in deployment or reuse. These files are particularly beneficial for handling large or complex data structures, including trained models from libraries such as $Scikit-learn$ or $XGBoost$, as they facilitate the storage and subsequent restoration of the entire object with minimal effort. Nonetheless, the *.pkl* format is specific to Python and encodes binary data, rendering it non-human-readable and unsuitable for cross-language use. When utilising .pkl files, it is essential to recognise the potential security risks associated with loading them from untrusted sources, as this may result in the execution of arbitrary code during the loading process. Consequently, it is advisable to utilise alternative formats such as *joblib* for extensive numerical data, *json* for fundamental objects, or more standardised and secure formats such as *ONNX* for cross-platform sharing of ML models. The generated artifacts are shown in Figure 4.12.

As our proposed DT covers the complete IEC continuum, we deployed the *.pkl* model to industrial edge layer and device, by addressing the low-latency and low

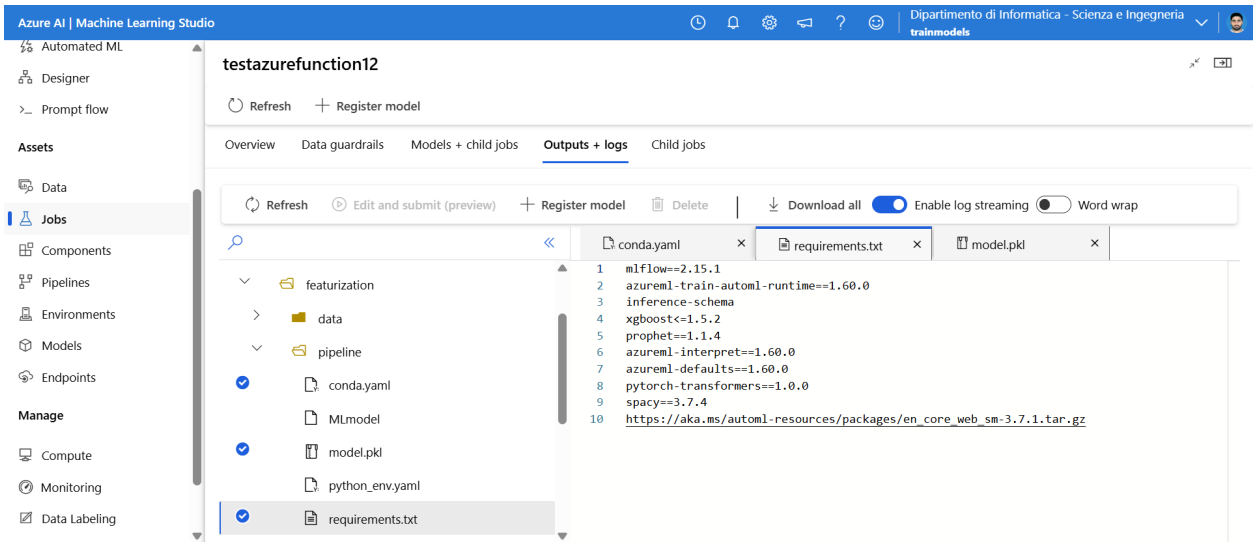

Figure 4.12: AutoML job artifacts list

computational power of edge devices.

4.3.4 Pkl Model Conversion to ONNX Standard

ONNX [128]is an open standard format for the representation of ML models. Co-developed by Microsoft and Facebook, it enables models learned in one framework (such as $PyTorch$, $TensorFlow$, $Scikit-learn$, or $XGBoost$) to be exported and executed in alternative environments or on diverse hardware. This adaptability is crucial for cross-platform implementation across cloud, edge, and embedded devices. ONNX facilitates hardware acceleration through the utilization of GPUs and specialized AI devices, enhancing inference speed and efficiency. As an open standard, it fosters consistency and reproducibility by providing a consistent model format that streamlines deployment, distinguishes between training and inference, and supports scaled production across many environments.

We converted the *.pkl* model into ONNX using its converter functions available in its library. Specifically, we leveraged the ONNXMLTOOLS [129] for ML model conversion. This Python package enables converting ML models developed in popular frameworks such as $Scikit-learn$, $XGBoost$, and $LightGBM$ to $ONNX$. The conversion process is not straightforward; first, we check the type of ML model trained with AutoML. The nature of the ML model changes when the ML task specification (Regression, Classification, and Time Series, etc) and the Dataset change during the AutoML job creation process. Therefore, we must use the relevant converter; otherwise, the conversion will fail. We had made a custom Python script for its conversion. Our industrial dataset has a regression problem, and given its features, our model is an XGBoost Regression pipeline. The conversion procedure is presented in bullet form below:

1. **Model Loading:** The process begins by loading a pre-trained ML model from a `.pkl` file using Python's `pickle` or `joblib` library. These libraries are commonly used to deserialize ML models saved to disk.

2. **Pipeline Verification:** A check is performed to determine whether the loaded

model is a part of an AutoML pipeline or a Scikit-learn pipeline. If the model is indeed a pipeline, the final step—typically the trained estimator—is extracted, as AutoML tools often encapsulate the core estimator within additional preprocessing and postprocessing components.

3. **Native Estimator Access:** In the case of *XGBoost*, the estimator is usually wrapped within a custom class. Therefore, the native XGBoost *booster* object is accessed using the `.model.get_booster()` method, which retrieves the underlying `Booster` model necessary for ONNX conversion.

4. **Input Specification:** Before conversion, the expected input format must be explicitly defined. This is accomplished using ONNX's `FloatTensorType`, which specifies the name, data type, and shape of the input tensor. This step ensures that the ONNX representation correctly interprets the model's input.

5. **Model Conversion:** The model is then converted into ONNX format using the *convert_xgboost* function provided by the `onnxmltools` library. This function takes as input the XGBoost booster and the previously defined input specification.

6. **Model Serialization:** The resulting ONNX model is serialized into binary format using the *SerializeToString*() method. It is then saved to disk as a portable `.onnx` file.

7. **Deployment Readiness:** The final ONNX model is now portable and suitable for deployment across a wide range of environments, including cloud infrastructures, edge devices, or systems built using different programming languages. It can be executed efficiently using the ONNX Runtime [130].

4.3.5 Future Recommendations on ONNX Conversion

During ML model-to-ONNX conversions, the converters are most often unavailable when the dataset is of a different type. Therefore, in the absence of a direct ML model converter to ONNX, various alternative methods may be employed based on the framework and deployment requirements. Initially, one must ascertain if the native framework includes inherent ONNX export capabilities; for instance, PyTorch provides torch.onnx.export, while TensorFlow facilitates model conversion to ONNX format using tf2onnx. Suppose the model is part of a framework lacking ONNX support. In that case, a feasible solution is to manually recreate the model architecture in a compatible framework, such as PyTorch or TensorFlow, retrain it on the original dataset and hyperparameters, and then export it using native tools. Alternatively, transforming the model into an intermediate format—such as CoreML or TensorFlow SavedModel—and subsequently converting that intermediate representation to ONNX using tools like *coremltools* [131] or *tf2onnx* [132] can be effective. When only the model's prediction interface is available and export is not feasible, the model can be encapsulated in a REST API or used as a black-box component with ONNX Runtime via custom extensions. Another technique involves producing an extensive array of inputs and outputs from the original model, then training a surrogate model within a supported framework that emulates its behavior and can be exported to ONNX. If these methods are ineffective, contacting the framework's developer

community or investigating GitHub repositories for experimental or community-supported converters may yield a solution. These solutions jointly ensure the feasibility of ONNX implementation without direct converters.

Addressing the RQ's, until now, we described how our proposed DT leverages AI technologies in the Cloud to access the industrial dataset, process it, transform it, and prepare it for training, and how automated training is performed. To support deployment of the artifacts, we reduced the pkl model size to the ONNX standard so it could be easily implemented on the industrial edge device. Recalling our proposed DT architecture in Figure 4.1, the vertical block description is now complete, including Data Processing, ML model training, and Model Selection. In the next phase, we will describe our architecture's Edge Deployment and Inference.

4.4 AI/ML Models Deployment at Industrial Edge and IoT Inference

The ultimate goal of converting an ML model to the ONNX format is its deployment on edge devices and also make it portable and adaptable to cross-platform devices. This section addresses RQ7. To start the implementation, we made a Docker image of the final ONNX model for inference. ONNXRUNTIME [130] is used for model inference on IoT devices. It is a high-performance, cross-platform inference engine designed to execute ML models stored in the ONNX format. It facilitates inference by enabling models learned in diverse frameworks—such as $PyTorch$, $TensorFlow$, $Scikit-learn$, or $XGBoost$ to be performed quickly across numerous contexts without requiring the original training framework. ONNX Runtime enhances the execution of these models for efficiency and resource use, facilitating both CPU and GPU acceleration, and enabling deployment on cloud, desktop, mobile, and edge devices. Utilizing ONNX Runtime allows developers to attain expedited inference times, diminished memory use, and enhanced portability of machine learning systems, rendering it optimal for production environments where consistency, performance, and scalability are paramount.

4.4.1 Containerization

It is the backbone of deploying the ML models on edge devices. The following points describe its importance in edge deployments:

- **Environmental Consistency**: Edge devices exhibit significant diversity in hardware, operating systems, and runtime environments. Containerization guarantees uniform model performance by encapsulating the model, its dependencies, and runtime environment into a singular, portable entity.
- **Security and Isolation**: Containers provide process-level isolation, mitigating the risk of conflicts and vulnerabilities. This is particularly crucial for edge devices that frequently constitute essential infrastructure or are linked to public networks.
- **Streamlined Dependency Management**: ML models generally depend on particular versions of libraries (e.g., Python, NumPy, ONNX Runtime).

Containers prevent version conflicts by encapsulating all dependencies, hence obviating the necessity for manual configuration on each device.

- **Scalability and Automation**: Containers facilitate scalable, automated deployments through orchestration systems like Azure IoT Edge or Kubernetes. Remote execution of updates, rollbacks, and monitoring is feasible without physical access to the edge devices.
- **Reproducibility and Portability**: Containerized models can be evaluated locally or in the cloud and subsequently deployed to production without alteration, guaranteeing uniform performance across various environments.
- **Compatibility with Edge Orchestration Platforms**: Contemporary edge platforms are engineered to deploy and administer containerized modules. Avoiding containerization would result in the forfeiture of capabilities such as remote monitoring, resource management, and automated failure recovery.
- **Operational Efficiency**: Containers optimize the entire edge deployment lifecycle by minimizing setup errors, facilitating automated rollouts, and simplifying troubleshooting.

4.4.2 C2E-AIOps Edge-IoT Deployment

By implementing containerization capabilities in our DT, we created a Docker image of a converted ONNX model and packaged it in a Dockerfile. This Dockerfile is created for inference on an industrial IoT device. It has all the necessary packages, such as *onnxruntime*, *paho-mqtt*, and *numpy* in its requirements.txt file. $Paho-MQTT$ is a Python package that offers a client class for publishing and subscribing to messages using the MQTT protocol. For data exchange between the industrial IoT device and the ONNX model, a widely used industrial communication protocol, MQTT, has been implemented. To enable real-time predictions in an IoT setting, we developed a lightweight, containerised inference pipeline using MQTT and ONNX. Firstly, we created an inference script written in Python that establishes an MQTT-based prediction system utilising an ONNX model to provide real-time predictions from incoming messages from IoT devices. The comprehensive procedure of our inference script is available in algorithm 2.

This script creates a comprehensive MQTT communication loop between an IoT data source and an ML prediction system utilising an ONNX model. The process begins with the import of necessary libraries: $json$ for parsing MQTT message payloads, *numpy* for managing numerical data arrays, *paho.mqtt.client* for facilitating MQTT communication, *onnxruntime* for executing ONNX model inference, and *hashlib* for transforming categorical string values into uniform numerical formats. The script subsequently establishes essential configuration parameters, including the MQTT broker(Mosquitto Broker) [133] address (BROKER), the port (1883), the topic for IoT devices to publish input data *(input/measurement)*, the topic for returning predictions *(output/predictk2001)*, and the file path to the ONNX model ($xgboostmodel\_mqtt\_prediction.onnx$). Subsequently, it initializes the ONNX model via $onnxruntime.InferenceSession$, retrieving the necessary input and output variable names for proper interaction with the model. The script also defines a utility function named. *hashstring* to process string-based categorical

Algorithm 2: Real-time MQTT-Based ONNX Model Prediction

Input:

incoming_msg: JSON sensor data from MQTT topic `"input/measurement"`

onnx_model: Pre-trained XGBoost model in ONNX format

Output:

prediction: Target variable prediction published to `"output/predict"`

Procedure *MAIN()*:

- *session* ← `ONNXRuntime.InferenceSession(onnxmodel)`;
- *client* ← `paho.Client(client_id)`;
- *client.connect*(`"mosquittobroker"`, 1883);
- *client.subscribe*(`"input/measurement"`);
- *client.on_message* ← `ON_MESSAGE_CALLBACK`;
- *client.loop_forever*();
- `/* Topic of communication security is deliberately ignored because the focus has been on aspects strictly related to the thesis (no TLS has been used). */`

Function *ON_MESSAGE_CALLBACK(msg)*:

- *data* ← `json.loads`(msg.payload);
- *vector* ← [ ];
- **foreach** *(field, value) in data* **do**
  - **if** *field is numeric* **then**
    - $norm\_val \leftarrow \frac{value - min\_range}{max\_range - min\_range}$;
    - *vector.append*(*norm_val*);
  - **else if** *field is categorical* **then**
    - *hashed* ← `hash`(value) % 1000;
    - *vector.append*(*hashed*/1000.0);
- *input_tensor* ← `np.array(vector).astype(np.float32)`;
- *prediction* ← `session.run(None, {"input": input_tensor})[0]`;
- *client.publish*(`"output/predict"`, str(prediction[0]));

data, such as device IDs or instrument names. This function hashes the input string using MD5, converts the hash to an integer, and normalises it to a float within the range of 0 to 1. This guarantees a uniform numerical representation of strings appropriate for input into ML models. The preparation logic is primarily executed in the $preprocess_input$ function, which takes a parsed JSON object and converts data into a NumPy array in the precise order and format required by the model. The function hashes categorical features such as $Instrument$, $Serial\_no$, and $deviceId$, while directly converting numerical features such as $Measurement\_attr$ and $Measurement\_value$ to floats. The payload is converted from bytes to a UTF-8 string and then parsed as a JSON object.

The script establishes an $on\_message()$ callback function that is triggered whenever a new MQTT message is received on the subscribed topic. The payload is converted from bytes to a UTF-8 string and subsequently parsed as a JSON object. The preprocessed features are input into the ONNX model, and the prediction outcome is obtained. The model output, usually a NumPy float, is subsequently transformed into a native Python float and encapsulated within a JSON object, which is then published to the output MQTT topic *(output/predictk2001)*. This enables downstream systems—such as dashboards, actuators, or warning systems—to receive and respond to the prediction results in real time. Additionally, this script establishes the MQTT client, designates the $on\_message$ function as a callback for incoming messages, connects to the broker, and subscribes to the input topic. $loop\_forever()$, which maintains the script's continuous operation, monitoring for messages and doing real-time inference. infinite loop utilising the client. The script's continuous operation involves monitoring for messages and performing real-time inference. The outcome of our script is a streamlined and effective real-time ML inference system suitable for deployment on edge devices or gateways, connecting IoT data with ML/AI model predictions via MQTT communication.

4.5 MLOps Workflow in our C2E-AIOps Framework

This section specifically addresses the RQ8 mentioned in Section 1.1. MLOps leverages Development Operations (DevOps) principles to effectively incorporate ML models into software systems. MLOps strives to integrate many ML stages, including model building, quality assurance, model delivery, and monitoring, into a single integrated process [134]. By standardising different development and operational procedures, it strives to keep an ML system reliable, strong, and reproducible [135, 136]. MLOps can bring together data scientists and IT experts by incorporating CI/CD pipelines into the development process, therefore streamlining ML deployment timelines [137]. MLOps facilitates a smooth transition of ML models from development to quality assurance by automating data and model validation in the CI phase. Furthermore, CD pipelines enable the reliable, rapid deployment of verified models to the production environment. The standard MLOps lifecycle is categorised into these essential phases [138].

1. **Data Ingestion**: This step involves aggregating data from multiple sources and transforming them into the formats necessary for ML models.

2. **Model Development and Training**: It refers to the designing of ML models by data scientists and ML engineers and the models' training using prepared datasets.

3. **Model Validation**: It involves testing and validating the ML model on unseen data, evaluating its performance, and estimating its accuracy in the real world.

4. **Experiment Tracking**: It facilitates the tracking and management of different ML experiments and allows the comparison between experiments to learn and interpret the impact of changes in hyperparameters on the model.

5. **Model Versioning**: This step keeps track of validated models and ensures traceability and accountability in ML workflows.

6. **Model Deployment**: Once the model is approved and ready for delivery, this step deploys the model to the production environment using CD pipelines.

7. **Model Monitoring**: It is essential to continuously monitor the model performance and detect any degradation due to data or concept drift. This pipeline step enables proper logging and monitoring for auditing and debugging purposes.

8. **Model Retraining**: Once the model's performance degrades below a predefined threshold, it is necessary to retrain the model using the latest data and invoke the MLOps workflow again.

The graphical representation of the MLOps lifecycle is seen in Figure 4.13.

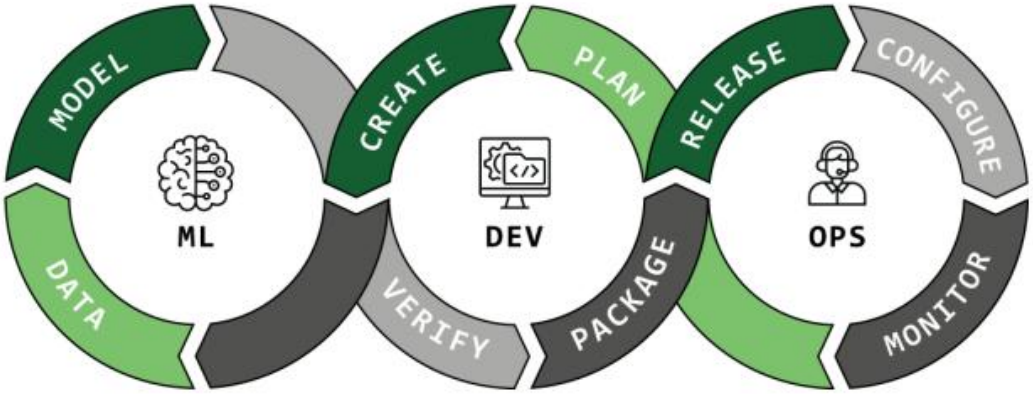

Figure 4.13: MLOps lifecycle

Many open-source and corporate MLOps technologies offer functionalities to optimise the ML life cycle. MLFlow, an open-source platform [139], provides functionalities of experiment tracking, model registry, and model serving [140]. Kubeflow [141] is a scalable method for deploying ML workflows within a Kubernetes environment, with each workflow step encapsulated in a container. TensorFlow Extended (TFX) [142], developed by Google, is another prominent MLOps solution that offers a framework for data validation, model training, and model serving. Amazon SageMaker, Azure Machine Learning, and Google Cloud Platform Vertex

AI are commercial cloud ML services that offer a variety of solutions across the ML lifecycle [143].

In our C2E-AIOps architecture, we exploited GitHub Actions with the Azure ML platform to implement MLOps best practices in our industrial DT. GitHub Actions are free, while Azure ML offers different pricing plans based on the services used. The detailed analysis of the cost of Azure ML services will be addressed in the next section of the dissertation. We addressed the challenges below by implementing MLOps best practices in our industrial scenario.

1. **Technical Challenge**: Designing MLOps systems to handle fluctuating demand, especially during ML training, remains a major hurdle.
2. **Organisational Challenges**:
    - Resistance to adopting the mindset and culture required for DS and ML practices.
    - Industrial stakeholders often take a product-focused approach, misaligned with ML system development.
    - Shortage of skilled professionals, including data engineers, ML engineers, DevOps engineers, and architects.
    - Lack of collaboration within multidisciplinary teams is needed for successful ML operations.
3. **Operational Challenges**:
    - Difficulty in manually operating ML systems due to the complex interaction of software and hardware components.
    - Resolving support requests is complicated by the involvement of multiple parties and components, with failures stemming from both ML infrastructure and software within the MLOps stack.

We automated our proposed industrial DT using GitHub Actions [124]. GitHub Actions workflow is started by creating a *.yml* file in a GitHub repository. The details of this automation pipeline are in the next section.

4.5.1 Our CI/CD Pipeline Workflow/Automation

In the previous section, we discussed the creation of a Docker image that has an ONNX model, a *requirements.txt* file, an inference script, and an MQTT configuration for the inference of the ONNX model with an industrial IoT device. Our CI/CD pipeline automates the Docker Image creation, pushes it to the industrial GitLab/Container registry, and updates its IoT Central edge Manifest and device with the latest ONNX model inference. Its workflow is described in algorithm 3: The following steps are involved in the implementation of our CI/CD workflow:

- When a new ONNX model file corresponding to the path models/*.onnx is pushed to the primary branch of our GitHub repository, the pipeline is automatically initiated. At the outset, the Docker image name and version, the Azure IoT Central API URL, the application and device identifiers, the manifest,

and the device template IDs are all globally defined via environment variables. A single task, *build* and *push*, is defined in the pipeline and executes on the most recent Ubuntu environment. Two critical secrets are securely accessed from the GitHub repository within this job: $AZURECR\_PAT$ for Docker registry authentication and $IOT\_CENTRAL\_TOKEN$ for IoT Central API authentication of the industrial instance.

- The GitHub repository code is initially processed through the standard actions/checkout@v3 action. The personal access token is then used to log in to a *GitLab* or $AZURE$ container registry. The Docker image is subsequently constructed from the current context (.), tagged with both a version-specific tag and a latest tag, and subsequently published to the registry. The jq and *curl* system dependencies are installed to support JSON processing and $HTTP$ requests, respectively, during Docker image construction.

- Subsequently, the pipeline sends a GET request (via curl) to Azure IoT Central to retrieve the current deployment manifest. This manifest is stored locally as $current\_manifest.json$. It then updates the version and image fields for the $onnxmqttinference$ module in the manifest using the jq tool, saving the output as $updated\_manifest.json$. This revised manifest is printed for debugging purposes and subsequently uploaded as a build artifact.

- The artifact is re-downloaded to the $./manifests$ folder, simulating a real-world scenario in which a different task or step could consume it. The pipeline then replaces the existing deployment configuration with the updated one by submitting a PUT request to update the deployment manifest in IoT Central. Lastly, the end-to-end deployment process is finalized by issuing a $POST$ request to simulate pushing the manifest to the IoT Edge device. The updated manifest is printed as a final step to verify that the correct image and version were applied to IoT Central.

Secrets help ensure credential security, eliminating the need to hardcode them into the pipeline. In our study, we used secret credentials for *GitLab*, *Azure* container registries, and $IoTCentral$ token. One pipeline handles everything, including updating ML models, building containers, and setting up devices. By automating these procedures, our proposed DT achieves real-time responsiveness, adaptability, and automation —essential for industrial applications where model drift and system downtime must be mitigated. The outcome is a closed-loop system in which physical assets and their digital equivalents progress simultaneously through versioned model updates and synchronised IoT device orchestration.

Algorithm 3: CI/CD Workflow for Docker Image Creation, Push to Registry, and Update Industrial IoT Central Edge Manifest

Trigger: On push of `onnx` model to `main` branch

Environment Variables:

`IMAGE_NAME`, `IMAGE_VERSION`, `IOT_CENTRAL_API`, `IOT_CENTRAL_APP_ID`
`DEVICE_ID`, `MANIFEST_ID`, `DEVICE_TEMPLATE_ID`
`AZURECR_PAT`, `IOT_CENTRAL_TOKEN`

Procedure *CI/CD Pipeline()*:

- Checkout the GitHub repository;
- Authenticate to GitLab Docker Registry using `AZURECR_PAT`;
- Build Docker image tagged with `IMAGE_VERSION` and `latest`;
- Push both Docker images to the registry;
- Install dependencies: `jq`, `curl`;
- Retrieve current deployment manifest from IoT Central via `GET` request;
- Save as `current_manifest.json`;
- Use `jq` to update fields:;
  - `onnxmqttinference.version` ← `IMAGE_VERSION`;
  - `onnxmqttinference.settings.image` ← `IMAGE_NAME:IMAGE_VERSION`;
- Save as `updated_manifest.json`;
- Upload `updated_manifest.json` as GitHub Actions artifact;
- Send `PUT` request to IoT Central with updated manifest;
- Send `POST` request to deploy manifest to target edge device;
- Print final updated manifest for verification;

Chapter 5

C2E-AIOps Implementation Details and Performance Results

5.1 System Details

We have used an industrial system to implement our C2E-AIOps framework. The details of this system are: an Intel Core i9-10900X @ 3.70GHz CPU, 32.0GB of RAM, a 2GB graphics card, and 6.39TB of disk storage.

5.2 Implementation and Automation Workflow of C2E-AIOps

The proposed framework was developed in collaboration with MARPOSS S.p.A. [44] as part of the development of the MAINDO product [144], to enhance MAINDO's prediction capabilities based on real production data. We have implemented the training procedure based on our industrial dataset in the Azure cloud using custom Python scripts and its services such as Azure ML/AI Studio, and AutoML (Azure ML SDK Version 1.60.0 and its sub-packages *azuremlcore*, *azuremltrain*, *azuremlautoml*). The developed codes are available at [145, 146]. Python version 3.9.21 is used in this implementation. The open-source package and environment management system *conda* has been used, particularly in Azure Functions implementations, to create a virtual environment. It resolved conflicts between virtual and global Python packages, including their versions. A specific version of the packages has been used in this implementation and is mentioned in the *requirements.txt* file. As we described our C2E-AIOps framework in detail in section 4.1.3. Now we discuss how we implemented it programmatically. So, we created a Jupyter Notebook written in Python. Jupyter Notebook [147] is a free, web-based interactive computing environment that lets developers make and share documents with live code, equations, graphs, and text that tells a story. It works with several programming languages, but Python is the most common. It is widely used in DS, ML, and academic research for tasks such as exploratory analysis, data visualization, model development, and reporting. It is an excellent tool for both learning and professional growth because it combines code execution, rich text, and multimedia features in a single document. You can run notebooks on your computer or on hosted platforms. This makes it easier to work together and achieve the same results across different locations.

Firstly, this Jupyter notebook initiates the AutoML process in Azure ML. An *Experiment* was established within an existing Workspace, which functions as the central repository for all ML resources. A compute target is created, which entails either building a new *AmlCompute* cluster or attaching an existing one to manage the model training workloads. Once the compute is ready, the dataset is organized by importing the data into a TabularDataset, which provides a structured, schema-aware view of the input data. The fundamental configuration for AutoML is defined by $AutoMLConfig$, which specifies parameters such as the job type (e.g., regression, classification), the target column, iteration limits, and evaluation metrics. This setup is encapsulated within an *AutoMLStep*, which integrates AutoML into a pipeline step, enabling scalable, reproducible ML workflows. The pipeline is subsequently submitted to train the model utilizing the designated *AmlCompute* target, so commencing the experiment run. Upon completion of training, outcomes can be examined via the experiment run history, metric visualizations, and a model performance scoreboard. The optimal model is evaluated by extracting it from the run artifacts and applying it to new or hold-out test data to assess its prediction efficacy. This optimized AutoML pipeline facilitates effective model construction, selection, and assessment. A single Jupyter Notebook handles this complete procedure. End-to-end industrial dataset processing, data transformation, ML model training using AutoML, artifact generation, and pkl model conversion to ONNX standards have all been performed by running a single notebook, as shown in Figures 4.2 to 4.13. Every AutoML job creates a unique ID, and we can troubleshoot or check any job using that ID. It is also crucial to retrain the ML model on a new dataset for a particular job ID.

Afterwards, we downloaded the artifacts. And checked the nature of the trained ML model. The trained ML model in our industrial scenario is shown in Figure 5.1.

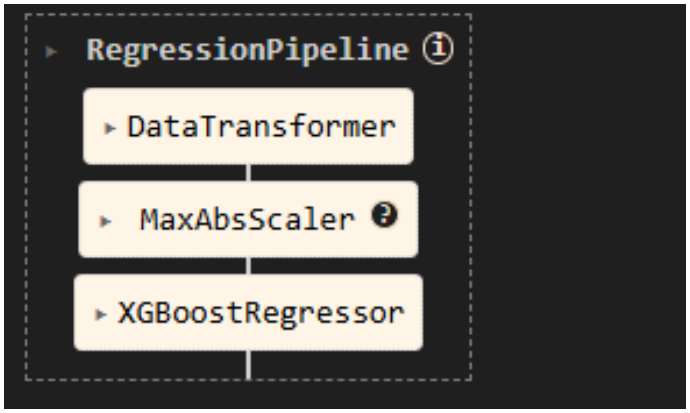

Figure 5.1: Type of trained ML model

In converting from pkl to onnx, the Notebook code first checks the model and generates the output shown in Figure 5.2, then checks the model's nature before converting it to ONNX format. As our script or code generated a pipeline-based model, our code verifies it, as shown in Figure 5.3.

After verification, our customized code converted the pipeline-based ML model into ONNX format. Please note that this conversion is based on the problem type specified during AutoML job creation and on the dataset attributes. So this conversion is customized. The conversion code demonstrates converting a trained *XGBoost* regression model to the ONNX format using the *onnxmltools*

```
# Load the AutoML trained model
with open(model_path, "rb") as f:
    automl_model = pickle.load(f)

print(f"Loaded model type: {type(automl_model)}")
```

```
Loaded model type: <class 'azureml.training.tabular.models.forecasting_pipeline_wrapper.RegressionPipeline'>
```

Figure 5.2: Checking ML model type

```
if hasattr(automl_model, "steps"):
    print("Model is a pipeline. Extracting the final step.")
    fitted_model = automl_model.steps[-1][1]  # Extract the last step (ML model)
else:
    print("Model is NOT a pipeline. Using it directly.")
    fitted_model = automl_model

print(f"Final model type: {type(fitted_model)}")
```

```
Model is a pipeline. Extracting the final step.
Final model type: <class 'azureml.automl.runtime.shared.model_wrappers.XGBoostRegressor'>
```

Figure 5.3: Verification of pipeline-based ML model

module, facilitating model portability across various contexts and frameworks. The procedure commences with the import of essential functions: $convert\_xgboost()$ for model transformation and $FloatTensorType$ to delineate the model's input schema. The code assumes that a native $XGBoost$ model ($native\_model$) has been previously trained and verifies the model type via a print statement. The model's input type is defined using $FloatTensorType$, indicating that it requires an input tensor with an indeterminate number of rows (None) and precisely five features (columns). The $convert\_xgboost()$ function is utilized to transform the model's booster object, obtained using $get\_booster()$, into the ONNX format, while providing the specified input schema. Upon conversion, the ONNX model is stored on disk as $xgboostmodel\_mqtt\_prediction.onnx$ utilizing binary write mode. A confirmation message is generated at the end to indicate a successful save; however, it contains a minor error: the variable name $onnx_model_path$ should be used within an f-string to display the file name accurately. This code is particularly advantageous for delivering ML models to platforms such as IoT devices or edge settings that use ONNX inference engines. The code snippet of conversion and its output are shown in Figure 5.4.

Similarly, this model can be saved in the Model registry of Azure ML/AI Studio. The registered ML and ONNX models in our Azure ML/AI studio workspace are shown in Figure 5.5 and can now be used for inference on industrial edge and IoT devices.

5.2.1 ONNX model Inference with Industrial Edge Device

In the next phase of implementation, we exploited the artifacts generated at the end of the AutoML job. Although the AutoML job generated a larger volume of artifact files and additional dependencies, we used containerization, as explained earlier in the dissertation, to reduce the size of artifact files and better utilize edge resources

```python
from onnxmltools.convert import convert_xgboost
from onnxmltools.convert.common.data_types import FloatTensorType

# This is your native XGBRegressor model (unwrapped from AutoML)
# Make sure it's trained
print(type(native_model))

# Define input type
initial_type = [("input", FloatTensorType([None, 5]))]

# Convert to ONNX
onnx_model = convert_xgboost(native_model.get_booster(), initial_types=initial_type)

# Save to file
onnx_model_path = "xgboostmodel_mqtt_prediction.onnx"
with open("xgboostmodel_mqtt_prediction.onnx", "wb") as f:
    f.write(onnx_model.SerializeToString())

print("ONNX model saved as {onnx_model_path}")

```

```
<class 'xgboost.sklearn.XGBRegressor'>
ONNX model saved as {onnx_model_path}
```

```python
onnx_model_path
```

```
'xgboostmodel_mqtt_prediction.onnx'
```

Figure 5.4: ONNX model converted successfully

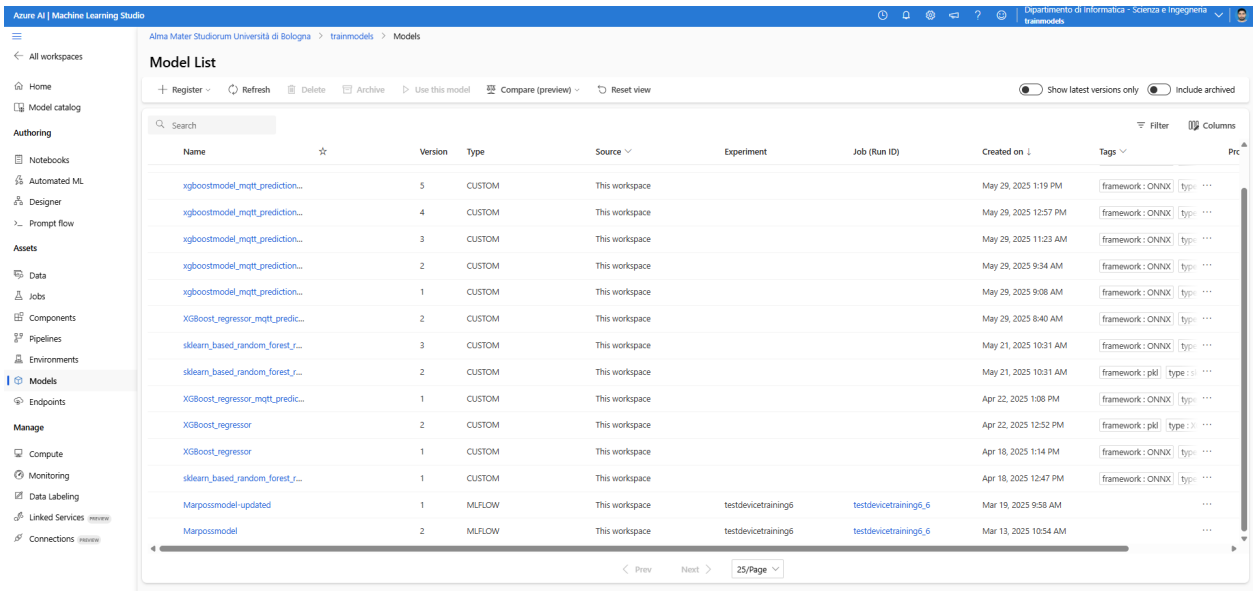

Figure 5.5: Registered ML and ONNX models

for our ONNX model inference. We use the *onnxmodel*, *requirements.txt*, and *inference.py* files to build a Docker image for our proposed C2E-AIOps framework. The implementation code is here [145]. The details of the inference script have already been discussed in section 4.1.5. An ONNX model can do predictions on an industrial Edge device. Our Python script creates an MQTT communication loop between an IoT data source and an ML prediction system using an ONNX model. It imports necessary libraries, establishes configuration parameters, and defines a utility function called *hashstring* to process string-based categorical data. This script also establishes an *on_message* callback function for incoming messages, converts the payload to a $UTF-8$ string, and parses it as a $JSON$ object. The preprocessed features are then input into the ONNX model, and the prediction outcome is obtained. The model output is converted to a native Python float and published to the output MQTT topic, enabling downstream systems to receive and respond to the prediction results instantly. The script also establishes the MQTT client, designates the *on_message*() function as the callback for incoming messages, connects to the broker, and subscribes to the input topic. The MQTT configuration used for ONNX model inference is shown in Figure 5.6. First, we tested the MQTT settings for inference using an ONNX model with the MQTTX tool [148]. We can see in Figure 5.6 that the incoming JSON message is parsed to the subscriber topic, and the prediction is generated, indicating that the ONNX model has started predicting. After this check, we implemented it on our industrial edge device.

```
# MQTT & Model Configuration
BROKER = "localhost"
PORT = 1883
SUBSCRIBE_TOPIC = "input/measurement"
PUBLISH_TOPIC = "output/predictk2001"
MODEL_PATH = "xgboostmodel_mqtt_prediction.onnx"
```

Figure 5.6: MQTT configurations

Notably, we leveraged the $ONNXRUNTIME$ [130] library. Without this library, inference is not possible. We used Linux as a subsystem on Windows OS. We created a Docker image on it using the *requirements.txt* file, the inference script that was described earlier, and an ONNX model. Docker image creation can be seen in Figure 5.7.

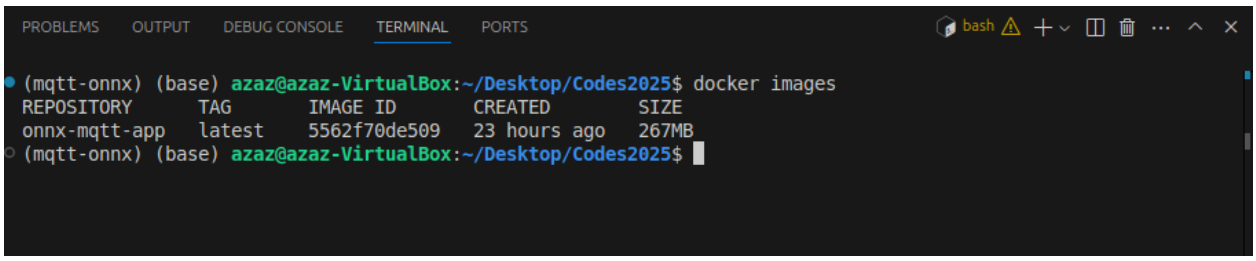

Figure 5.7: Docekr Image creation for ONNX model inference

5.2.2 Real-Time on Device Deployment and Prediction

After creating the Docker image, we deployed the ONNX model on a real industrial device. When we run a Dockerfile, a container is built and started. So we made and started a container of our Docker image. It can be seen in Figure 5.8. By running this container, we started inference on the ONNX model with the MQTT protocol. Mosquitto MQTT broker [133] has been used in a Linux terminal to publish the prediction. The input is provided in JSON format and is predicted based on the publishing topic. To start the inference, we first started the Mosquitto broker service in the Linux terminal. It can be seen in Figure 5.9. An input message in a $JSON$ format has been sent to the publishing topic $input/measurement$ using MQTT local host configurations as described in algorithm 2. $mosquitto\_pub$ command has been used to publish the message/input on the $ONNXRUNTIME$ for inference. The publishing procedure is illustrated in Figure 5.10.

Figure 5.8: Containerization of our Docker Image

Figure 5.9: Mosquito Broker client initiation

Similarly, the $mosquitto\_sub$ command enables us to receive the output results of the ONNX model using ONNXRUNTIME and subscriber topic $output/predictK2001$. When we run this command, it gives the predictions on the ONNX model. Predictions can be seen in Figure 5.11.

5.2.3 CI/CD Implementation

Our CI/CD pipeline process started when our trained ONNX model was committed to our GitHub repository. We created a GitHub repository earlier for this and a Python script that is already built for creating an AutoML job, generating a trained

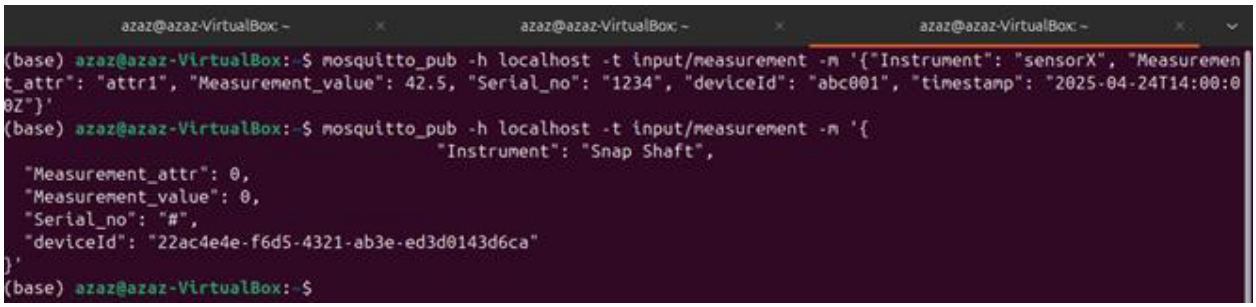

Figure 5.10: Mosquito Broker client publishing to ONNXRUNITME

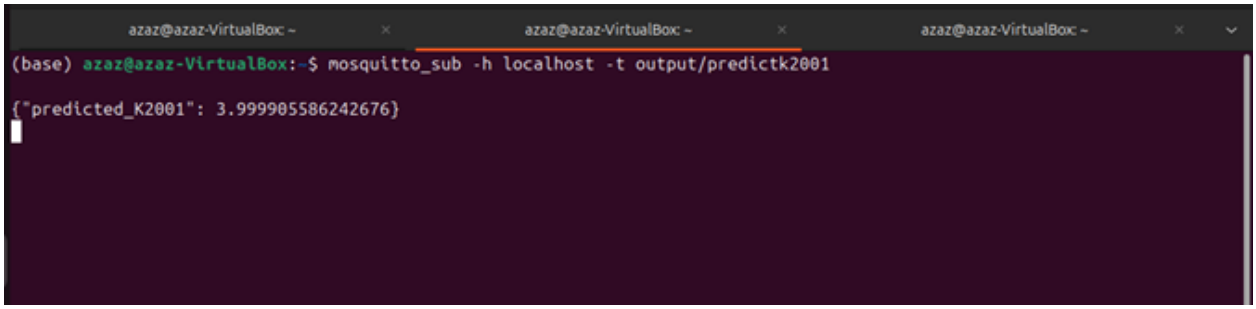

Figure 5.11: Mosquito Broker client subscribing for ONNX model predictions

ML model, and converting it into ONNX. At the end of this script, a custom function is made with the name *commit_model_to_github*(). This function uses the GitHub REST API to automatically commit an ONNX model file that is stored on our local disk to our GitHub repository. This function is made to interact with cloud-based processes like Azure Functions. It makes sure that ML models are always in the correct version and that their deployment is tracked. When the code runs, it first imports important libraries like *os*, *base*64, and *requests*. It then gets credentials that are specific to the context, such as the GitHub access token $GITHUB\_TOKEN$, which is used for secure authentication. The function sets metadata like the name of the repository, the owner of the repository, the target branch (main), and the file path that the file should take inside the repository.

Next, the ONNX model is loaded from the local file system in binary mode, and its contents are encoded in *Base*64 format so that they may be uploaded to GitHub's API. The function builds the correct GitHub API URL to interact with the file location in the repository that you want to work with, and it uses an HTTP *GET* request to check if the file currently exists. If so, it gets the SHA identifier for the file, which is needed to update existing files instead of making new ones. Then, a JSON payload is created that has the commit message, the encoded file content, the destination branch, and the SHA if it is needed. We provided this payload to the GitHub API using a *PUT* request. The function certifies that the commit was successful if the request was successful, as indicated by status codes 200 or 201. If it doesn't work, it prints the HTTP status code and error message that are needed for debugging. Lastly, the function sends back a textual message saying that the model commit procedure is done.

This function provides a strong and automated way to push model artifacts in a GitHub repository. This helps make ML development processes more reproducible and traceable. This function implementation procedure is also described in algorithm

4.

Algorithm 4: Commit ONNX Model to GitHub Repository for CI/CD

Input: GitHub token stored as environment variable (`GITHUB_TOKEN`), local model path M_{local}, GitHub repo owner O, repo name R, branch B, and GitHub file path P

Output: A confirmation message indicating success or failure of commit

Read GitHub token: `token` ← `getenv("GITHUB_TOKEN")`;
Set repo metadata: *owner* ← O, *repo* ← R, *branch* ← B;
Set file paths: *local_path* ← M_{local}, *target_path* ← P;
Read model content in binary: *content* ← `read(local_path)`;
Encode content in Base64: *encoded* ← `Base64Encode(content)`;
Build API URL: `url` ← `"https://api.github.com/repos/" + owner + "/" + repo + "/contents/" + target_path`;
Set HTTP headers with Authorization and Accept fields;
Send GET request to `url` to check if file exists;
if *response status code = 200* **then**
    Retrieve file SHA from JSON response: *sha* ← `response["sha"]`;
else
    sha ← `None`;
Create commit payload with `message`, `content = encoded`, `branch = B`;
if *sha* ≠ *None* **then**
    Add `sha` to payload for file update;
Send PUT request with payload to `url`;
if *status code* ∈ {200, 201} **then**
    return `"Model committed to GitHub successfully!"`;
else
    return `"Failed to commit: " + status code`;

After the model was successfully committed, our $YAML$ pipeline workflow began, as described in section 4.1.6. The GitHub Actions Workflow began after the ONNX model was pushed to the main branch of GitHub repository. We can see the status of our workflow runs on GitHub Actions. It works according to the instructions mentioned in the $YAML$ file. The final results are in figures 5.12 and 5.13. After this step, our latest Docker image with the latest ONNX model is uploaded to the industrial edge device. Our Docker image manifest implemented on an industrial use case is shown in List 5.1. Our containerised Docker solution was successfully deployed on the industrial edge device. This is also evident from Figure 5.14.

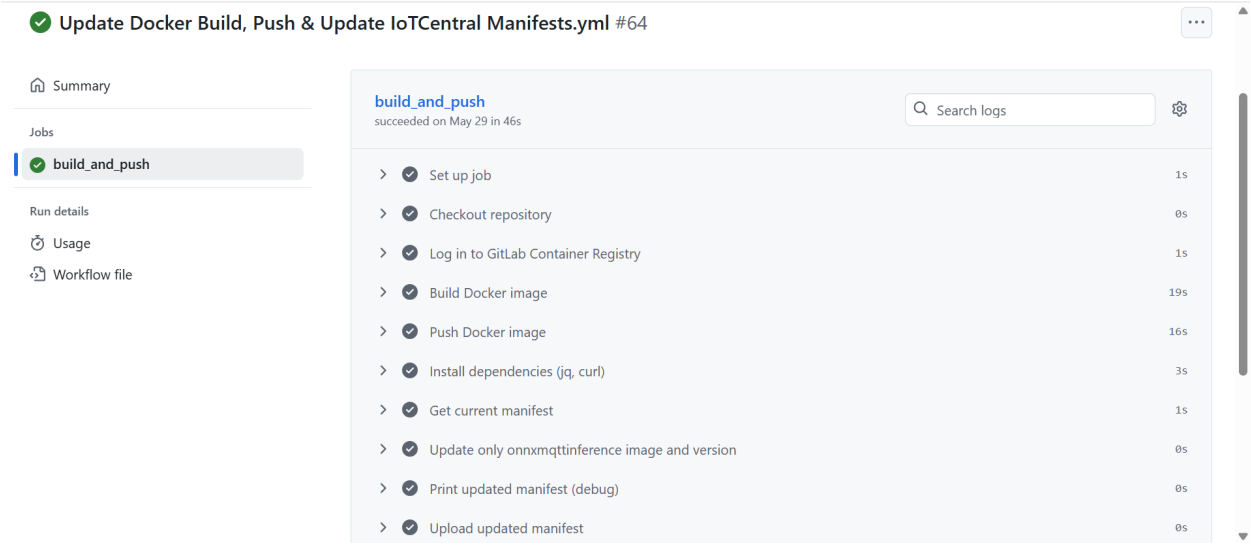

Figure 5.12: GitHub Actions workflow results(1)

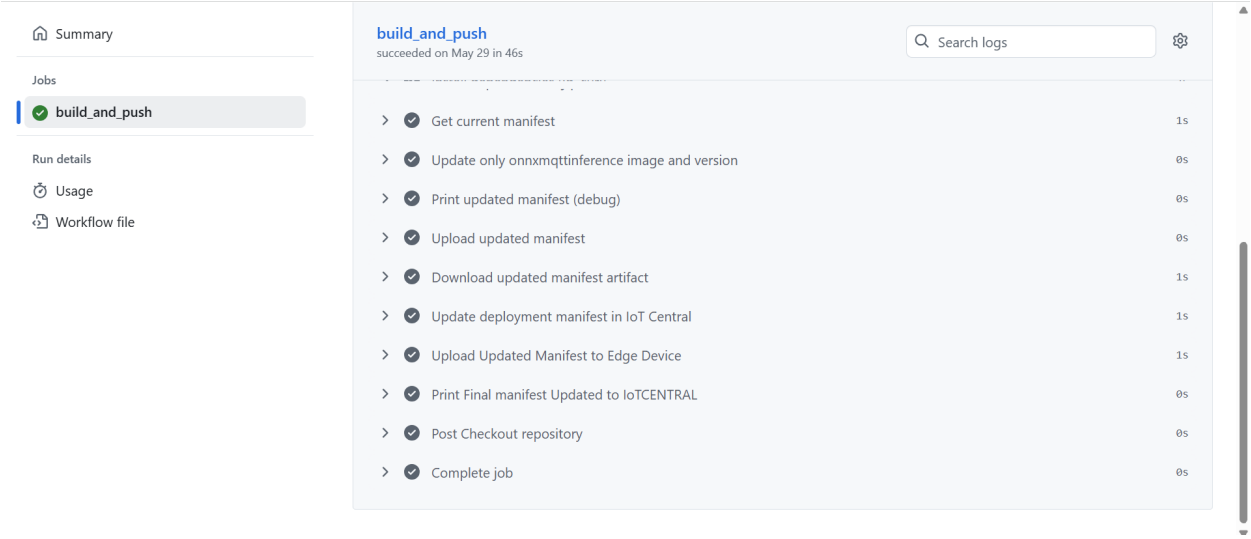

Figure 5.13: GitHub Actions workflow results(2)

```
{
  "onnxmqttinference": {
    "version": "1.6.0",
    "type": "docker",
    "status": "running",
    "restartPolicy": "always",
    "startupOrder": 6,
    "settings": {
      "image": "azaz-onnx-with-mqtt-predict:1.6.0",
      "createOptions": "{}"
    },
    "env": {
      "MQTT_BROKER": {
        "value": "mosquittobroker"
      },
      "MQTT_PORT": {
        "value": 1883
      },
      "MQTT_KEEPALIVE": {
        "value": 60
      },
      "MQTT_SUBSCRIBE_TOPIC": {
        "value": "input/measurement"
      },
      "MQTT_PUBLISH_TOPIC": {
        "value": "output/predictk2001"
      },
      "MQTT_ONNX_MODEL_PATH": {
        "value": "xgboostmodel_mqtt_prediction.onnx"
      }
    }
  }
}
```

Listing 5.1: Our Docker Image manifest on Marposs edge deployment manifest

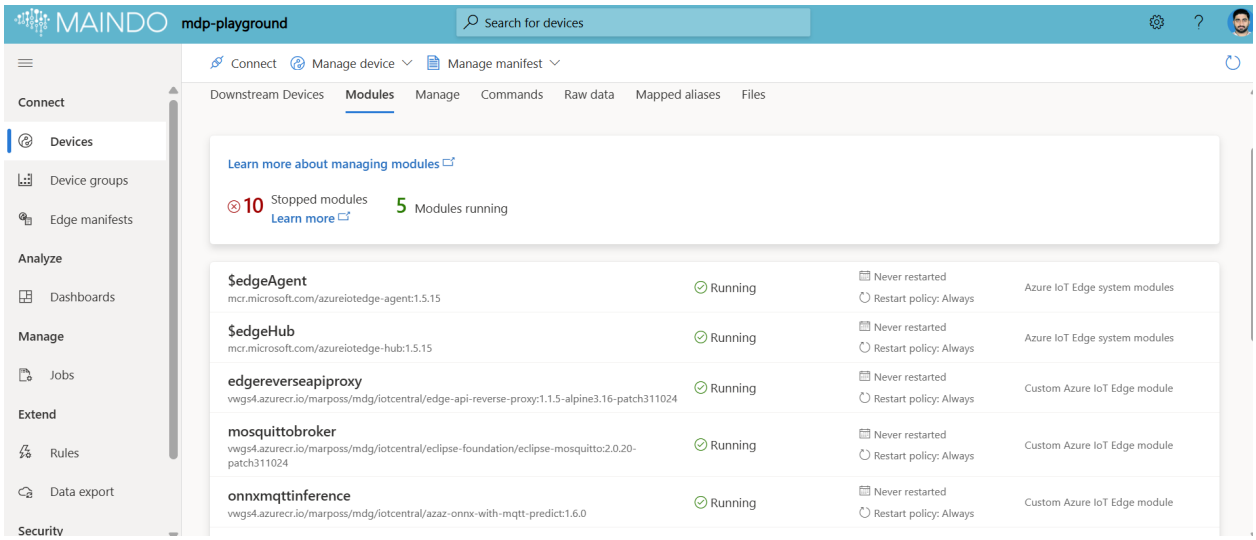

Figure 5.14: Our Docker solution running on a Marposs edge device

5.3 Azure Function Deployment–Automation of our C2E-AIOps Framework

The objective of this study is also to implement and explore emerging technologies within an industrial system, addressing the growing needs of industry. This is why, in our previous sections of the dissertation, we exploited the latest technologies. Similarly, we also implemented Azure Functions [45] in our C2E-AIOps framework to some extent. Azure Functions are essential in Industry 4.0, enabling event-driven, serverless computing tailored to the evolving requirements of smart manufacturing settings. They enable industries to execute specific logic or microservices in response to triggers such as sensor data, changes in machine status, or system alarms, without the need for infrastructure management. This serverless methodology [45]guarantees scalability, cost-effectiveness, and adaptability, particularly in settings with variable workloads. Azure Functions effortlessly interact with industrial systems such as MES, SCADA, and IoT platforms, facilitating real-time data processing and communication between edge and cloud components. They play a crucial role in managing ML pipelines and updating DT models, hence improving predictive maintenance and operational intelligence. Azure Functions enable agile, intelligent automation in Industry 4.0 ecosystems with robust security, automated scaling, and seamless integration with existing workflows. It is a paid service based on the application you want to create and is available through the MS Azure Subscription plan. The cost details have been discussed in the Implementation section of this dissertation, specifically in the cost Analysis part.

We encapsulated all our Python scripts and customized functions in an Azure Function. When Azure function started through CLI, Azure CLI or any other terminal, it started the AutoML job, downloaded the artifacts, converted the ML model to ONNX standard, and committed it to the GitHub repo where an already built CI/CD Pipeline retrieve the current IoT Central manifest, build the Docker image, pushed it to industrial container registry, updated the industrial IoT Central edge manifest and device with latest Docker image of ONNX model and perform predictions. All these steps have been done by just triggering the Azure Function

once. When an Azure Function starts, it triggers a URL call, and after the process completes, it returns an HTTP status code indicating whether the execution was successful. The cost of execution depends on the subscription plan you choose, which includes Pay-as-you-go, Premium, Webservice, or Containerised. The pricing details can be found here: [149]

First, we created an Azure Function in our local industrial system using VS Code and the Git Bash terminal. We installed the Azure Function extension in VS Code and then started it with the *funcstart* command. To connect it with Azure ML/AI Studio services, a configuration file must be present in its project directory. This configuration file is in *JSON* format and has our MS AZURE *subscription_id*, *resource_group*, and *workspace_name*. We placed our configuration file in its project directory—a Python script file, *automl_logic.py*, which encapsulates all the customised functions discussed above in our dissertation, is also placed in the Azure Functions project directory. For confidentiality reasons, its implementation is currently private but can be found here: [150]. The Azure Function project directory structure is shown in Figure 5.15. It also has *config.json*, *requirements.txt*, and an ONNX model. Azure Function triggers each time on a fixed URL, as shown in Figure 5.16. Its project folder also has a primary function called *function_app.py* in addition to *automl_logic.py*. When we started the Azure Function via the CLI, *function_app.py* executed *automl_logic.py*, which became a single solution for our C2E-AIOps.

function_app.py is in the TriggerAutoML directory, and its code snippet can be seen in Figure 5.17. *function_app.py* defines a serverless function triggered by *HTTP* that automates an ML workflow, with an emphasis on training and deploying models using cloud-native services. When the function receives an *HTTP* request, it starts a predefined AutoML pipeline by calling the *run_automl_logic*() function. This function trains a model using Azure ML services and converts it to the ONNX format. Then, the *commit_model_to_github*() method runs, enabling easy integration with version control by automatically uploading the updated model to our GitHub repository. Try-except blocks are used to handle errors properly, ensuring that runtime failures are logged correctly and returned with the appropriate *HTTP* response codes, as described in detail in the next section. Overall, the function has a lightweight, flexible, and scalable design that works well in situations where ML models are retrained and updated regularly in response to new data or user-driven events.

This single Azure Function automated all the operations of our proposed C2E-AIOps framework. We received a trigger 200 (OK) message when all operations were performed successfully, and the same results were generated as in our previous customized scripts. Ultimately, through a single CLI command, *funcstart*, a containerized version of our C2E-AIOps framework is deployed on the industrial edge device, enabling efficient anomaly prediction. Let's also discuss the error handling mechanism of Azure Functions in the following section.

5.3.1 Azure Function- Troubleshooting

Status codes are essential in Azure Functions with HTTP triggers for conveying the results of function execution to the client [151]. A 200 (OK) status code signifies that the function executed correctly and yielded a valid result. If the client submits

Name	Type	Compressed size	Password pr...	Size	Ratio
.vscode	File folder				
artifacts	File folder				
azureml	File folder				
TriggerAutoML	File folder				
.funcignore	FUNCIGNORE File	1 KB	No	1 KB	0%
.gitignore	Git Ignore Source File	1 KB	No	1 KB	37%
automl_logic	Python Source File	4 KB	No	11 KB	68%
AzuriteConfig	File	1 KB	No	1 KB	0%
config	JSON Source File	1 KB	No	1 KB	15%
function_app	Python Source File	1 KB	No	1 KB	26%
getting_started	Markdown Source File	1 KB	No	3 KB	64%
host	JSON Source File	1 KB	No	1 KB	0%
LICENSE	File	1 KB	No	2 KB	41%
README	Markdown Source File	1 KB	No	1 KB	0%
requirements	Text Document	1 KB	No	1 KB	28%
xgboostmodel_mqtt_prediction.onnx	ONNX File	2 KB	No	6 KB	81%

Figure 5.15: Azure Function project directory

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE: ACTIVITY LOG        host start - Task

Functions:

        HttpExample: [GET,POST] http://localhost:7071/api/HttpExample

For detailed output, run func with --verbose flag.
[2022-05-25T18:45:53.470Z] Worker process started and initialized.
[2022-05-25T18:45:54.960Z] Host lock lease acquired by instance ID '00000000000000000000000E24CCCAC'.
```

Figure 5.16: Azure Function triggered

```python
import logging
import azure.functions as func
from automl_logic import run_automl_logic
from automl_logic import commit_model_to_github

def main(req: func.HttpRequest) -> func.HttpResponse:
    try:
        logging.info("Running AutoML logic and committing onnx model to GitHub...")
        run_automl_logic()
        commit_model_to_github()
        logging.info("Model conversion and commit succeeded.")
        return func.HttpResponse("Automation Script and Model commit executed successfully.", status_code=200)
    except Exception as e:
        logging.error(f"Error: {str(e)}")
        return func.HttpResponse(f"Internal ServerError occurred : {str(e)}", status_code=500)
```

Figure 5.17: Main function to trigger Azure Function

faulty information or omits requisite arguments, the function may return a 400 (Bad Request) status code, indicating that the problem lies in the request content. When permission fails due to missing or incorrect function keys or tokens, the function generally returns a 401 (Unauthorised) or 403 (Forbidden) status, depending on the security settings. A 500 (Internal Server Error) signifies that an error occurred during execution, maybe due to a runtime exception or improper output binding configuration. These status codes are crucial for troubleshooting and for developing resilient client-side programs that can respond appropriately to various results. The tabular view with description of these status codes is in Table 5.1.

Table 5.1: Summary of common HTTP status codes in Azure Functions

Code	Status	Meaning	Common Causes
200	OK	Function executed successfully	Valid input received; logic completed without errors
400	Bad Request	Client-side input error	Missing or malformed input; invalid JSON; incorrect headers or parameters
401	Unauthorized	Missing or invalid authentication key	Function key or token not provided, expired, or not recognized
403	Forbidden	Insufficient permissions	Key provided but lacks privileges; IP/network restrictions
404	Not Found	Function or route not found	Incorrect URL or route; unsupported HTTP method for the function
500	Internal Server Error	Execution failure	Unhandled exceptions; misconfigured bindings; logic errors in function code
503	Service-Unavailable	Temporary issue or downtime	Cold start latency; throttling; function app not ready

5.3.2 Cost Analysis

As our C2E-AIOps leverages cloud-based technologies, we will discuss the costs for each service used during the framework implementation here. First, we have an MS Azure Subscription account owned by the ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA. We have been assigned a Resource group in which we use cloud services. The services are: *Azure IoT Central*, *Blob Storage*, *Azure ML/AI Studio*, and *Azure Function*. In Azure ML/AI Studio, we exploited *AutoML*, *Compute*, *Models*, *Data*, *Jobs*, *Pipelines*, *Environments*, and *Notebooks* services. The workspace name created to use Azure ML/AI Studio services was *trainmodels*. Multiple workspaces could be created in a single subscription. Every AutoML job incurs costs because we have created a computing machine to train our ML models. This cost is subjective because you have multiple options for clusters, like small-scale CPUs or even GPUs, depending on industry requirements and budget. In our case,

our compute cluster costs us $0.32/hr$ (when running), and one AutoML job took 10 to 12 minutes. We also used a *Blob Storage* account to receive data from industrial IoT devices. As mentioned earlier, Azure ML Studio is a complete service that provides multiple resources, as highlighted at the start of this section. In this service, we tested some jobs that were not completed successfully, so we avoided this cost in our analysis and reported only the price used for the successful operations. Cost associated with particular services can be seen in table 5.2

Table 5.2: Implementation Cost Estimation for the C2E-AIOps Framework

Resource	**Cost (€)**
Azure ML Service	192.00
Container Registry	14.77
Blob Storage	76.61
Azure Functions	48.75
GitHub Actions	0.00
Total	**332.13**

We used the Azure Pricing calculator [152] to estimate the cost of the services. The *Azure ML service* cost us 192 euros because it used Virtual Machines to compute the ML models. Similarly, a *Container Registry service* has been used to store the Docker images created for inference of our ONNX model on the company's edge and IoT devices, and it cost us 14.77 euros until the complete implementation of our C2E-AIOps framework. The *Blob Storage* account associated with the *IoTCentral* and *Azure ML service* costs us 76.61 euros. It also saved the best model artifacts and matrices generated by the AutoML service. We also used *Azure Functions* to deploy our C2E-AIOps as a serverless service, and it cost us 48.75 Euros. *GitHub Actions* were also used in this study, but they are very flexible and free of the cost associated with GitHub [153]. So our CI/CD pipeline solution is very lightweight and bears no additional cost as an Azure DevOps pipeline.

The annual cost to implement the C2E-AIOps framework is approximately €1.535. It also depends on the amount of data used during implementation, as large datasets or records require significant storage space and incur higher costs. Regarding Azure Function implementation, there are different subscription plans. If we opt for a containerized app plan, the exact annual fee will apply. However, if we use flexible plans like pay-as-you-go, the price may be less or more. The container registry cost is also included in this approximation, as we store the Docker images of our ONNX model and inference script within it. This projection cost could be a good fit for further discussion, allowing the C2E-AIOps to be integrated into a production-ready system.

Chapter 6

FedAdapt-CAD: Federated Learning Adaptation via Edge/Client Integration and Dynamic Model Adaptation

Ensuring data privacy at the edge during ML model training is a strategic imperative rather than a mere technical concern. Industrial data, including sensor readings, production statistics, and equipment health information, constitutes valuable intellectual property and process expertise that must be safeguarded against unauthorized access. By maintaining data at the edge, firms mitigate the risk of exposing sensitive information through cloud transfers, thereby enhancing cybersecurity and reducing vulnerability to data breaches. This method also facilitates adherence to stringent legal frameworks (e.g., GDPR [154] and ISO standards [155]) that regulate data management in industrial settings. In addition to regulatory and security issues, edge-based privacy methods protect a company's competitive edge, as operational data frequently contains secret techniques and efficiency models [156]. Moreover, in collaborative Industry 4.0 ecosystems with multiple stakeholders, privacy-preserving edge training cultivates trust by preventing any partner from obtaining unauthorised access to another's datasets. This paradigm offers dual advantages: it safeguards sensitive industrial information while minimising latency and bandwidth requirements, enabling real-time responsiveness and data confidentiality in dispersed industrial systems. In this context, we implemented custom FL techniques using publicly available datasets. The evaluation results extracted establish a baseline, allowing us to integrate them effectively into our C2E-AIOps framework in the future.

This chapter offers a comprehensive introduction to FL, emphasising its core principles. Also, we will address RQ3 by proposing a novel FL framework, FedAdapt-CAD, designed explicitly for real-world industrial datasets and applicable to other benchmark datasets as well. It has two main components. The first is Client-Aware Aggregation (CAA), and the second is Dynamic Model Adaptation (DMA). The following section will present a detailed discussion of these contributions.

6.1 FL Introduction

FL is a new distributed ML framework that facilitates collaborative training of a shared ML model across multiple clients, such as mobile devices, sensors, or institutions, without the direct exchange of their data [157]. This paradigm guarantees that raw data is retained locally on each client, with only model updates or gradients sent to a central server for aggregation. This method mitigates privacy issues, lowers communication expenses, and facilitates collaborative learning across geographically dispersed datasets [158]. FL has attracted much attention in recent years owing to its potential to transform industries. FL was initially launched by Google in 2016 [127] as a method to enhance mobile applications by training predictive models across millions of devices without the need to collect raw data. Nonetheless, its potential applications extend well beyond mobile devices, including healthcare, banking, and the IoT, where data privacy is paramount [159].

Figure 6.1 demonstrates that FL (b) differentiates itself from conventional centralized ML (a) in multiple aspects. The latter strategy involves collecting data from various sources and consolidating it on a central server for model training, raising substantial concerns about data privacy and security, especially in sensitive fields regulated by stringent laws, such as the GDPR [160]. Conversely, FL relies on decentralized data management, enabling model training directly on local devices or data repositories. The procedure commences with the aggregator initializing a global model using designated learning parameters, which each client subsequently downloads. Clients subsequently calculate model updates using their local datasets and transmit them to the aggregator.

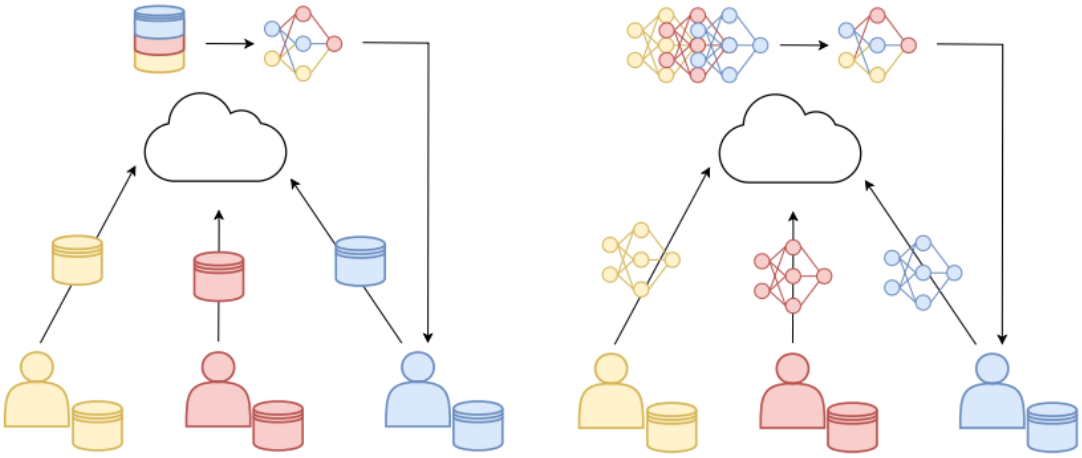

Figure 6.1: Differences between ML(a) and FL (b) approaches

The aggregator consolidates these local changes to improve the global model, using diverse aggregation algorithms based on the nature of the updates received. In the 2016 Google proposal, [127] presented Federated SGD (FedSGD), an algorithm that utilizes Stochastic Gradient Descent (SGD) for federated optimization. During each training iteration, clients perform a single SGD step on the current model and send the computed gradients to the server. They proposed Federated Averaging

(FedAVG), a variant in which clients transmit model weights rather than gradients, facilitating several local updates before transmission and improving efficiency.

This recurrent process persists as clients download the enhanced global model, compute additional updates, and repeat until global training concludes. Additionally, novel aggregation strategies have been created to improve model training across varied client datasets. One such example is FedProx [41], which incorporates a proximal term during aggregation to mitigate issues arising from data heterogeneity. FedProx promotes adherence to the global model by penalizing deviations, hence enhancing convergence and resilience among customers. The SCAFFOLD algorithm [161] represents a significant progression in this field, utilizing control variates to stabilize and improve the aggregation of updates from clients with non-IID data distributions. SCAFFOLD employs local control variates to modify updates from each client, thereby reducing variance and improving the consistency of contributions to the global model. This approach not only expedites convergence but also improves the resilience of the training procedure. This distributed architecture of FL substantially reduces the risk of disclosing sensitive information, as only model updates are transmitted to the central server, thereby safeguarding privacy and maintaining regulatory compliance. Most FL strategies depend on synchronous aggregation, which updates the global model only after receiving contributions from all participating clients. However, recent advancements in asynchronous approaches [162] enable continuous updates to the global model, allowing it to be sent back to clients for further training rounds even if some have not yet contributed.

6.2 Motivation

The increasing interconnectivity of industrial systems and the rapid proliferation of smart devices have led to massive volumes of data being generated across domains such as predictive maintenance, smart manufacturing, and AD. In Industry 4.0, ensuring secure and efficient AD is crucial, especially in distributed environments where privacy constraints prevent the sharing of raw data. FL offers a promising solution by enabling collaborative model training without transferring raw data [163, 164]. Instead, edge devices train local models and share only model updates with the central server, preserving privacy. However, applying FL to real-world industrial scenarios presents several challenges. Unlike general-purpose datasets, industrial data is often non-IID, high-dimensional, and collected under varying conditions from heterogeneous devices [165, 166].

In addition, industrial systems are typically structured into hierarchical layers—edge devices, gateways, and cloud infrastructure—that further complicate federated deployments. Data from such systems often involves streamed input, missing values, and other practical issues that degrade model convergence and accuracy. While some solutions have emerged through Hierarchical FL (HFL) frameworks [167, 168], many assume balanced data distribution or stable communication links—conditions that rarely hold in operational industrial environments. Resource constraints at edge devices, including limited computational power and memory, exacerbate the situation, hindering efficient local training and timely model updates [169]. Moreover, industrial workloads are dynamic, with devices joining or leaving the network and system configurations evolving. Current frameworks often fail to adapt to such changes, resulting in poor generalisation and scalability.

To address these concerns, this dissertation proposes a novel FL framework—**FedAdapt-CAD**—that integrates adaptive aggregation and dynamic learning strategies to better align with industrial deployment needs. Our approach generalises beyond specific datasets by addressing key industrial challenges, including streamed data input, frequent missing values, non-IID data, and resource constraints at the edge. Existing FL approaches for AD typically use fixed aggregation strategies (like *FedAvg*, *FedProx*, and *FedH2L)* or static hyperparameter settings. Our proposed FL framework directly combines CAA and DMA and addresses two core challenges—data heterogeneity and temporal drift in attack behaviours. This makes the proposed framework particularly well-suited for industrial environments where both client reliability and evolving threats are prominent concerns. This dual adaptation mechanism forms the foundation of this chapter contributions, which are summarised as follows:

1. The proposed CAA strategy enhances convergence and fairness in non-IID settings by weighing client updates based on local data quality and model performance.

2. DMA is a server-level hyperparameter tuning mechanism that dynamically adapts to changes in data distribution and attack patterns, ensuring model robustness and longevity.

3. The proposed framework is evaluated on three real-world industrial datasets—WADI [170], CMAPSS [171], and BATADAL [172]—demonstrating significant improvements in accuracy, recall, and convergence speed over existing FL solutions.

6.3 Related Work

FL enables collaborative model training without sharing raw data, making it suitable for privacy-sensitive industrial settings. However, challenges like data heterogeneity, resource constraints, and hierarchical infrastructure complicate its deployment.

Zhao et al. [173] proposed allocating a small fraction of globally accessible data (e.g., publicly available data) to customers' devices. Yao et al. [174] collected metadata contributed by volunteers to enable controllable meta-updating. Yoshida et al. [175] incentivized clients to provide local datasets and introduced a hybrid learning technique in which the server enhances the model by leveraging both the shared data and the clients' local models. These methodologies resulted in a significant improvement in FL accuracy; however, they raised concerns about potential data leakage due to the need to share clients' datasets. Moreover, publicly accessible datasets are not consistently available, particularly in domains where data is highly sensitive. Li et al. [163] proposed a client selection strategy to enhance robustness in heterogeneous industrial IoT data environments. Similarly, Liu et al. [165] conducted a comprehensive survey that highlighted non-IID data as a significant bottleneck to FL performance. In the context of vertical FL, Li et al. [164] designed a framework tailored for heterogeneous industrial regions that focuses more on vertical data splits than on hierarchical architecture. Dutta et al. [176] introduced a privacy-preserving FL framework for chemical engineering applications, emphasising cross-organisational collaboration. To handle the challenges of system hierarchy, several hierarchical FL (HFL) frameworks have been proposed. Fang et al. [166] developed a gradient correction mechanism for HFL with multi-timescale updates.

Huang et al. [177] presented a fairness-aware FL scheme suitable for hierarchical structures. Architectures such as client-edge-cloud HFL have also been investigated by Liu et al. [167] and Wang et al. [168], while FedH2L [169] focused on heterogeneous learning for resource-constrained IIoT environments. These works, however, often assume ideal network conditions or overlook the need for dynamic reconfiguration.

Moreover, frameworks such as those discussed in [178–181] offer improvements in scalability and collaborative intelligence but often lack mechanisms for real-time model adaptation and efficient edge-side optimization. The works of Zhou et al. [182] and Kang et al. [183] have explored FL applications in cyber-physical and critical infrastructure systems, but without addressing the fine-grained client-aware weighting or adaptive tuning proposed in our work. Yao et al. [184] emphasized that the global model encompasses greater global knowledge and should be retained as a reference rather than discarded. Consequently, they devised a two-stream model to convey global knowledge to the local model. Minimizing the maximum mean discrepancy (MMD) loss enables the two-stream model to extract more generalized information and acquire superior local representations. Subsequently, in [185], they employed 1×1 convolution, vector-weighted average, and scalar-weighted average operators to integrate global and local information. Li et al. [41] indicated that excessive local updates may lead to divergence in FL training, particularly in non-IID data scenarios. Consequently, they incorporated a proximal penalty term into the local objective loss functions to restrict the local model's divergence from the global model. Rieger et al. [186] noted that clients articulate representations in various ways, and that their collective knowledge may be obscured during synchronization. They also implemented conditional gated activation units to allow clients to condition their units. This enables clients to ascertain the expression of the global characteristic and adjust the global pattern accordingly. These methodologies necessitated increased storage capacity and computational demands on resource-limited client-side devices, resulting in more resource allocation and elevated energy usage.

Yeganeh et al. [187] integrated clients' local models by employing an adaptive weighting method that used the inverse distance between the local and averaged model parameters. This strategy will diminish the influence of out-of-distribution models, resulting in a global model with reduced variance. The authors additionally examined the interplay with other parameters, including training accuracy and data size. Nonetheless, these variants performed worse in our studies, perhaps due to excessive suppression of specific legitimate but "out-of-distribution" devices. Conversely, motivated by the capacity of momentum accumulation to mitigate oscillations [188], Hsu et al. [189] implemented the momentum optimizer on the server side and noted a substantial enhancement in FL accuracy. Subsequently, Reddi et al. [190] presented three sophisticated adaptive optimizers (namely, Adagrad [191], Adam [192], and Yogi [193]) to enhance the server-side global model. These adaptive federated optimizers enabled the use of adaptive learning rates across different gradients and achieved significant success; however, they required careful adjustment of initial learning rates, and we observed substantial fluctuations in accuracy throughout the experiments.

To overcome the limitations of previous framework-based FL approaches, our proposed framework introduces two novel components: *CAA* and *DMA*. *CAA* is a dynamic aggregation strategy at the client level that improves convergence and fairness in non-IID settings. And the DMA mechanism adapts to changes in data

distribution and cyber attack patterns at the server level. The framework is rigorously evaluated across three real-world industrial datasets —WADI, CMAPSS, and BATADAL —that span a range of industrial AD scenarios.

6.4 FedAdapt-CAD Framework Description and Workflow

The proposed FL framework is designed to address the inherent challenges of deploying FL in industrial environments. Two main components of this proposed system are CAA and DMA. To better understand their role and how they work, let us start by introducing our architecture (depicted in Fig. 6.2). The overall architecture consists of three main components: edge devices, regional aggregators, and a central server. Edge devices, such as industrial IoT sensors or embedded systems, perform local training on private datasets. These devices are grouped into multiple regions, each managed by a regional aggregator that collects and processes local model updates. The final aggregation of these regional models takes place at the central server to produce a global model.

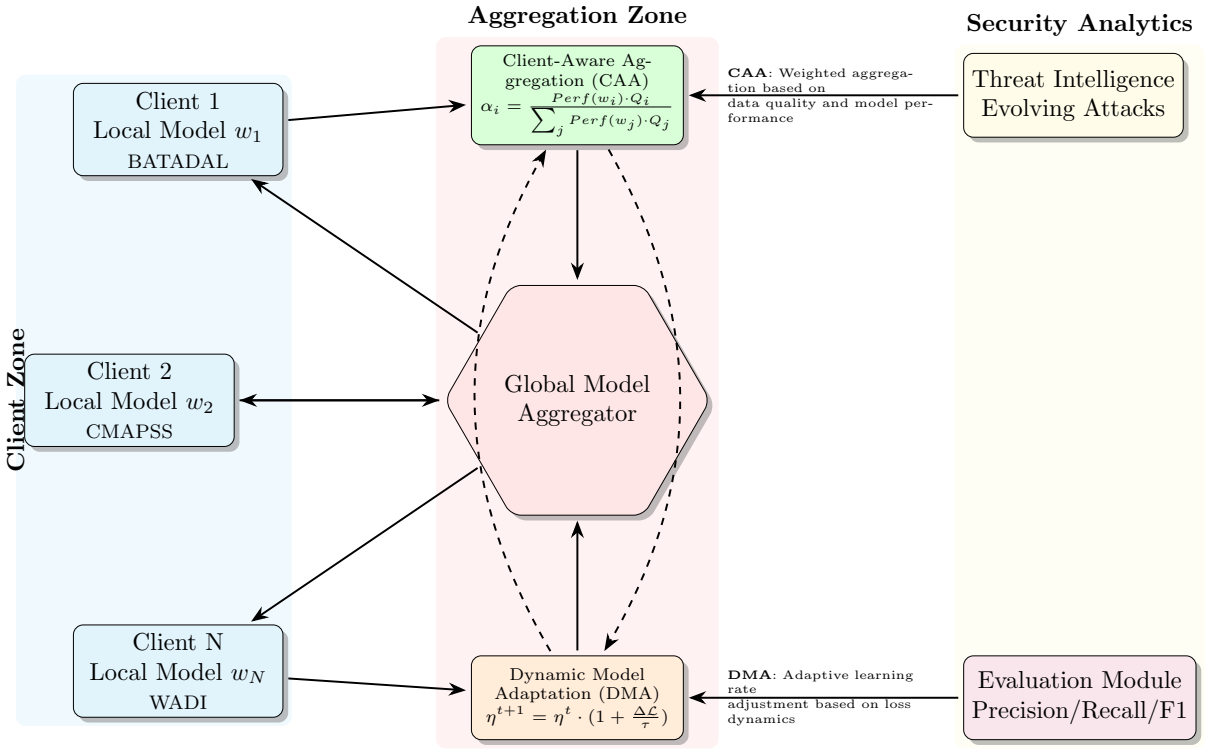

Figure 6.2: Proposed FedAdapt-CAD framework

6.4.1 Local Model Training

In the proposed framework, we consider a distributed learning scenario with N edge devices. These devices are geographically partitioned into M regions, where each region R_m contains n_m devices. By construction, the total number of devices across

all regions satisfies the condition:

$$\sum_{m=1}^{M} n_m = N. \tag{6.1}$$

Each device i in region m is assumed to hold its private dataset, denoted as $\mathcal{D}_{i,m}$. Importantly, these datasets are non-IID, meaning that each device's data follows a unique probability distribution $P_{i,m}(x, y)$. This data heterogeneity represents a fundamental challenge for collaborative learning, as it can significantly impact model convergence and overall performance. Therefore, adaptive aggregation strategies are essential for mitigating biases introduced by uneven or skewed data distributions. The ultimate goal of the system is to train a global model that minimises a collective loss function across all regions and devices. This global objective is defined as a weighted sum of the local loss functions:

$$\min_{w} F(w) = \sum_{m=1}^{M} \frac{n_m}{N} \sum_{i \in R_m} \frac{|\mathcal{D}_{i,m}|}{|\mathcal{D}_m|} F_{i,m}(w), \tag{6.2}$$

where w denotes the shared model parameters, and $|\mathcal{D}_m| = \sum_{i \in R_m} |\mathcal{D}_{i,m}|$ represents the total dataset size within region m. Here, each device's contribution is weighted by the proportion of its dataset within its region. In contrast, each region's contribution is further weighted by its fraction of the global device population. This hierarchical weighting ensures that both intra-region and inter-region data imbalances are systematically accounted for in the aggregation process.

The local loss function $F_{i,m}(w)$ for device i in region m is formally expressed as:

$$F_{i,m}(w) = \frac{1}{|\mathcal{D}_{i,m}|} \sum_{(x,y) \in \mathcal{D}_{i,m}} \ell(f_w(x), y), \tag{6.3}$$

where $f_w(x)$ denotes the output of the global model with parameters w, and y represents the ground-truth label. The loss function $\ell(\cdot, \cdot)$ quantifies the discrepancy between the model's prediction and the true label (for example, cross-entropy loss in classification tasks). In brief, we can say that this formulation captures the hierarchical and heterogeneous nature of the system in the following ways:

- Devices are grouped into regions.
- Each device contributes data of varying size and distribution.
- The global training objective carefully balances these contributions.

This structured design is essential to developing aggregation algorithms that remain robust under non-IID data conditions, a defining characteristic of real-world federated edge learning environments.

6.4.2 Regional Aggregation

In our hierarchical FL framework, each region is assigned a *regional aggregator* that serves as an intermediate node between the local devices and the global server. The role of the regional aggregator is to collect, aggregate, and summarise the models

trained by the edge devices within its corresponding region R_m. This approach reduces communication overhead with the global server and provides an additional layer of structure to address intra-region heterogeneity in device data. Formally, let $w_{i,m}^t$ denote the model parameters obtained from device i in region m after the t-th local training round. Since devices within the same area often contribute datasets of different sizes, their influence on the regional model must be weighted accordingly. The weighted aggregation ensures that devices with larger datasets, which typically contain more representative information, exert a proportionally greater impact on the regional model than devices with smaller datasets.

The regional model for region m is therefore updated at round $t+1$ according to:

$$w_m^{t+1} = \frac{\sum_{i \in R_m} |\mathcal{D}_{i,m}| \, w_{i,m}^t}{\sum_{i \in R_m} |\mathcal{D}_{i,m}|}, \tag{6.4}$$

where $|\mathcal{D}_{i,m}|$ represents the size of the dataset available at device i in region m. The numerator accumulates the weighted contributions of all devices in the region, while the denominator normalises these contributions to ensure that the aggregation produces a convex combination of local models.

This formulation closely resembles the classical *Federated Averaging (FedAvg)* strategy but applied at the regional level. However, the key distinction lies in its hierarchical structure. Instead of transmitting all device updates directly to the global server, regional aggregation consolidates information within each region before sending it to the global server. This two-level hierarchy offers several benefits:

- **Communication efficiency:** By aggregating locally within each region, the number of direct communication links to the global server is reduced from N devices to M regional models, thereby lowering network congestion and bandwidth requirements.
- **Scalability:** The framework naturally scales to large networks, as new devices can be added within a region without significantly increasing the global communication cost.
- **Heterogeneity mitigation:** Regional aggregation allows the system to handle non-IID data distributions better. Since devices in the same region often exhibit more similar data patterns (e.g., due to geographic or contextual similarity), regional aggregation produces more representative intermediate models before global integration.
- **Fault tolerance:** By introducing an intermediate aggregation layer, the system becomes more resilient to individual device dropouts or unreliable connections, as the regional aggregator still receives contributions from other devices in the same region.

Thus, this regional aggregation acts as a critical mechanism in hierarchical federated learning. It balances device contributions according to their dataset sizes, reduces communication costs, and improves robustness against data heterogeneity and network instability. The resulting regional models w_m^{t+1} serve as the building blocks for the subsequent global aggregation stage, where information from all regions is combined to update the global model.

6.4.3 Global Aggregation

Once the regional models have been computed, the next step in the hierarchical FL framework is the global aggregation performed at the central server. In this stage, the central server collects the regional models $\{w_m^t\}_{m=1}^M$ and combines them into a single global model w^{t+1} that will be shared with all devices for the next training round. This ensures that knowledge obtained across different regions is integrated into a unified representation of the system-wide learning task.

The global aggregation is defined as:

$$w^{t+1} = \frac{\sum_{m=1}^{M} n_m w_m^t}{N}, \tag{6.5}$$

where w_m^t denotes the model of region m at iteration t, n_m is the number of devices in region m, and $N = \sum_{m=1}^{M} n_m$ is the total number of devices across all regions. This formulation assigns each regional model a weight proportional to the number of devices in that region. Regions with more devices exert greater influence on the global model, reflecting the intuition that such regions typically account for a larger share of the overall data distribution. Conversely, regions with fewer devices have a smaller, but still meaningful, contribution. In this way, the aggregation ensures that the global model reflects the collective knowledge of the entire network while preventing the overrepresentation of any single region.

The global aggregation process is conceptually similar to the FedAvg algorithm but operates at the level of regional models rather than individual devices. This hierarchical approach provides several distinct advantages:

- **Efficiency in communication:** Instead of collecting updates from all N devices, the server only needs to aggregate M regional models, significantly reducing communication costs.
- **Fairness and balance:** By weighting regions according to their device population, the aggregation respects the relative scale of data contributions while avoiding bias towards regions with disproportionately large or small datasets.
- **Robustness to heterogeneity:** Since each regional model already integrates intra-region variations through regional aggregation, the global model is built upon more stable and representative building blocks, enhancing convergence in heterogeneous environments.
- **Scalability:** The two-level aggregation strategy makes the framework capable of supporting vast networks of devices distributed across multiple regions without overwhelming the central server.

In summary, this global aggregation step serves as the final integration stage in hierarchical FL. By combining regional models into a unified global model through weighted averaging, the system effectively captures knowledge across all devices and regions while maintaining efficiency, fairness, and robustness. The resulting global model w^{t+1} is then redistributed back to the regions and devices, initiating the next training cycle.

6.4.4 Adaptive Weighting for Non-IID Data

In industrial environments, edge devices frequently collect highly heterogeneous, non-IID data. Such heterogeneity can negatively affect the convergence and generalisation of FL models, as devices with skewed or limited datasets may disproportionately influence regional or global aggregation. To address this issue, an adaptive weighting mechanism is introduced that dynamically adjusts each device's contribution to its regional model based on the statistical properties of its local dataset. Specifically, each device i in region m is assigned a weight $\alpha_{i,m}$, which is computed using the variance of its dataset:

$$\alpha_{i,m} = \frac{\mathrm{Var}(\mathcal{D}_{i,m})}{\sum_{j \in R_m} \mathrm{Var}(\mathcal{D}_{j,m})}, \quad w_m^{t+1} = \sum_{i \in R_m} \alpha_{i,m} w_{i,m}^t. \tag{6.6}$$

Here, $\mathrm{Var}(\mathcal{D}_{i,m})$ represents the statistical variance of the dataset $\mathcal{D}_{i,m}$ belonging to device i in region m. The variance is a key indicator of the diversity and representativeness of the data samples:

- A **higher variance** suggests that the dataset covers a broader range of data patterns, making it more informative and valuable for training. Devices with higher-variance datasets are therefore assigned a larger weight in the aggregation process.
- A **lower variance** indicates that the dataset may be more homogeneous or limited in scope. Hence, its contribution to the regional model is scaled down to avoid biasing the overall learning process.

This adaptive weighting mechanism offers several benefits:

- **Mitigation of non-IID effects:** By accounting for data variance, the aggregation process reduces the risk of overfitting to narrow or skewed datasets, thereby improving generalisation.
- **Data-informed contributions:** Instead of relying solely on dataset size as a weighting factor, the approach incorporates data diversity, ensuring that more representative datasets have greater influence.
- **Fairness across devices:** Devices with small but diverse datasets are not overshadowed by devices with large but homogeneous data, promoting a more balanced aggregation.

In summary, our adaptive weighting refines the regional aggregation stage by incorporating dataset variance into the weighting scheme. This enables the system to handle heterogeneous, non-IID industrial data more effectively, thereby improving the stability and accuracy of the FL process.

6.5 Dynamic Model Adaptation (DMA)

In FL, for industrial systems, the global model is typically trained under the assumption that the underlying data distribution is stationary. However, real-world industrial environments are inherently dynamic and often experience *concept drift*,

in which the statistical properties of the input data evolve. Examples include the emergence of new attack signatures in the WADI dataset, sensor degradation and drift in CMAPSS, or evolving cyber-physical attacks in BATADAL. In such cases, a static global model with fixed hyperparameters often fails to generalise, leading to degraded performance. To overcome this limitation, we introduce *Dynamic Model Adaptation (DMA)*, a server-side mechanism that continuously optimises the model's hyperparameters in response to evolving data distributions and attack patterns.

The central idea of DMA is to treat key hyperparameters, such as the learning rate η and regularisation coefficients, as *adaptive and learnable entities* rather than static values. This approach enables the global model to self-adjust without requiring manual tuning, thereby improving robustness against non-stationary threats. The adaptation process takes place after each global aggregation round t, where the server leverages a held-out validation set $\mathcal{V}$ containing recent anomaly instances to guide hyperparameter updates. Formally, after the t-th round of global aggregation, the server computes a meta-gradient with respect to the learning rate:

$$\Delta\eta = \beta \cdot \nabla_{\eta} \mathcal{L}_{\text{val}}(w^t; \mathcal{V}), \tag{6.7}$$

where β is a meta-learning rate that controls the speed of adaptation, $\mathcal{L}_{\text{val}}$ denotes the validation loss, and w^t are the model parameters at round t. This meta-gradient quantifies how sensitive the validation loss is to the current learning rate, thereby providing a principled update direction. The learning rate is then updated as:

$$\eta^{t+1} = \eta^t - \Delta\eta. \tag{6.8}$$

This gradient-based adjustment mechanism ensures that the model prioritizes capturing recent anomalies and attack patterns while retaining knowledge from historical data. Unlike static FL frameworks, which rely on fixed optimisation settings throughout training, DMA introduces a feedback loop between model performance on recent data and the hyperparameters that govern the optimisation process. In doing so, DMA explicitly addresses temporal concept drift by dynamically aligning the training dynamics with the evolving industrial environment. The benefits of DMA are summarised as: **Adaptation to non-stationary environments:** By continuously updating hyperparameters, DMA enables the global model to remain effective in the face of evolving attack patterns, sensor drift, and other sources of data distribution shift. **Automation of hyperparameter tuning:** DMA eliminates the need for manual reconfiguration of learning rates or regularisation parameters, reducing operational overhead in large-scale industrial systems. **Meta-learning inspired optimisation:** By treating hyperparameters as learnable quantities, DMA integrates principles of meta-learning into federated learning, resulting in a more flexible and adaptive training pipeline. **Improved resilience to anomalies:** Since the validation set $\mathcal{V}$ is constructed from recent anomaly instances, the model adapts quickly to emerging threats while still preserving general knowledge from past training rounds.

Briefly, our DMA component introduces a meta-learning-based mechanism for hyperparameter optimisation in FL. By continuously adjusting hyperparameters such as the learning rate based on recent validation feedback, DMA ensures that the global model remains adaptive, resilient, and effective in dynamic industrial environments where non-stationarity and concept drift are unavoidable. Our proposed approach

focuses on data-driven client weighting and temporal hyperparameter adaptation, distinguishing it from conventional FL methods and addressing both spatial (non-IID) and temporal (concept drift) challenges in industrial systems.

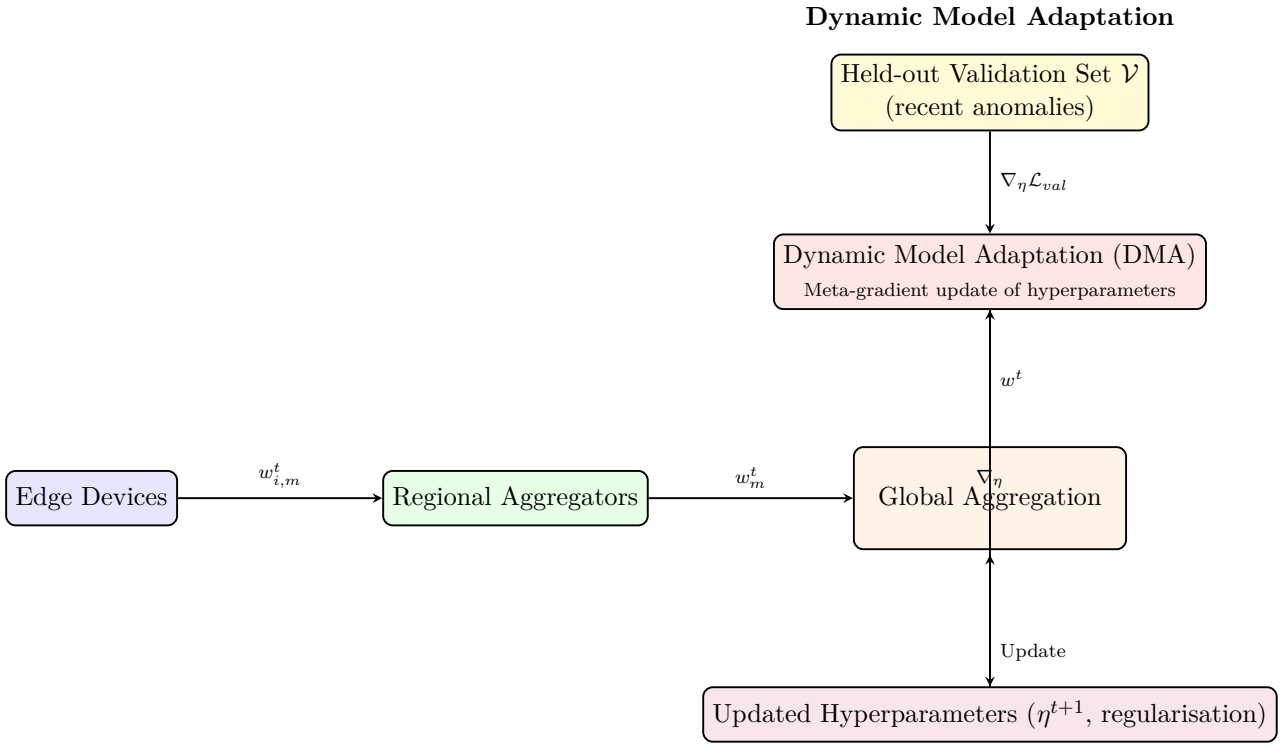

Figure 6.3: Illustration of the Dynamic Model Adaptation (DMA) mechanism. The server uses a validation set of recent anomalies to compute a meta-gradient, dynamically updating hyperparameters such as the learning rate η. This allows the global model to adapt to evolving industrial data distributions and attack patterns.

6.5.1 Federated Training Loop

FedAdapt-CAD employs a hierarchical *client-edge-cloud* architecture, intended to support extensive industrial systems with geographically dispersed devices. The training process is conducted in repeated communication rounds, in which models are progressively modified through local updates, regional aggregation, and global integration. Each communication round follows this sequence:

1. **Local Training:** Edge devices, also known as clients, do model training on their local datasets. A resource-aware optimization technique is used to address limited computing resources and memory constraints. This technique adaptively modifies hyperparameters, including the batch size and learning rate, based on the device's current hardware capacity. By tailoring the training process to each device's specific capabilities, the framework ensures efficient resource utilization while preserving high-quality model updates. Local training produces a collection of device-specific model parameters $w^t_{i,m}$, which capture information from the device's non-IID dataset.

2. **Regional Aggregation:** After local training, devices within the same geographic or logical area transmit their trained models to a regional aggregator.

The aggregator consolidates multiple models into a single regional model w_m^{t+1} via weighted averaging, with each device's contribution proportional to the magnitude or quality of its dataset, potentially modified by adaptive weighting techniques such as dataset variance. This intermediate aggregation decreases communication overhead to the central server. It mitigates intra-regional data heterogeneity by generating a representative regional model that encapsulates the collective knowledge of all devices within the region.

3. **Global Aggregation:** The central server aggregates regional models from all areas to derive the updated global model w^{t+1} via a second-level aggregation. In this phase, each regional model is weighted by the number of devices it represents, ensuring that regions with more contributing clients exert a proportionately greater influence on the global model. This hierarchical aggregation technique optimizes scalability, equity, and resilience: it minimizes direct connections to individual devices, mitigates inter-region heterogeneity, and maintains the contributions of all areas in proportion to their scale.

4. **Model Distribution:** Upon updating the global model, it is redistributed to all clients via the regional aggregators. Each regional aggregator transmits the revised model to its associated devices, allowing them to commence the subsequent round of local training using the latest global insights. This feedback loop ensures ongoing model improvement across successive iterations and enables the system to progressively adapt to changing data distributions, temporal concept drift, and new anomalies in industrial settings.

This federated training loop creates an organized process that combines local computation, regional summarization, and global model synthesis. The system utilizes hierarchical aggregation and resource-aware local optimization to attain efficient training, scalable communication, and resilient learning in highly heterogeneous and dynamic industrial data environments.

6.6 Datasets, Implementation Insights, and Performance Results

This section presents the experimental results obtained using FedAdapt-CAD on the CMAPSS, BATADAL, and WADI datasets. The evaluation focuses on model accuracy, error metrics, feature importance, and federated model performance across different clients. We employ a lightweight feedforward neural network for FL training, with ReLU activations and a softmax output layer. Hyperparameters such as learning rate, batch size, and local epochs were dynamically adjusted via the proposed DMA module. Unlike standard FL, our adaptivity mechanism enables real-time tuning based on client performance and global validation loss. This adaptivity leads to measurable gains in convergence speed and recall, particularly under non-IID settings.

6.6.1 Dataset Description

Table 6.1 summarizes the primary characteristics of the considered datasets.

Table 6.1: Summary of Industrial Datasets

Dataset	Domain	Samples	Features
WADI	Water Distribution System	784,571	124
CMAPSS	Aircraft Engine Health	49,986	26
BATADAL	AD	50,000	43

WADI Dataset

It comprises 784,571 samples with 124 numerical features and is derived from real-world water distribution system sensor readings under normal and attack scenarios.

CMAPSS Dataset

It comprises 49,986 operational aircraft engine samples, each labelled with Remaining Useful Life (RUL) and classified into discrete classes for classification tasks.

BATADAL Dataset

It consists of 50,000 samples with 43 features, focuses on AD in industrial systems, and includes preprocessing steps such as standardisation and time-series analysis.

6.7 Implementation Details

We have implemented our FedAdapt-CAD prototype in Python by using TensorFlow 2.0. Federated training uses multiprocessing to mimic real-world parallelism. NVIDIA RTX GPUs are used for server-side computations, while edge-device constraints are emulated by using Docker containers with throttled resources (RAM and CPU). The federated coordination logic, client orchestration, and secure communication between nodes are managed using the Flower framework. For the CMAPSS dataset, Mean Squared Error (MSE) is used as the primary loss function during training, as it penalises significant prediction errors. Mean Absolute Error (MAE) and the R^2 score are reported for evaluation—MAE reflects average error magnitude, while R^2 indicates how well predictions match the actual values (with 1.0 being a perfect fit). Together, these metrics assess both training performance and post-training regression quality.

6.8 CMAPSS Dataset Results

6.8.1 Global Model Performance

Table 6.2 summarises the performance of the global federated model on the CMAPSS dataset in terms of Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 score. The global model achieves a significantly lower MSE and a higher R^2 score than local client models, indicating improved generalisation.

Table 6.2: Global Model Performance on CMAPSS Dataset

Metric	**Global Model Value**
MSE	190.806
MAE	9.653
R^2 Score	0.9598

6.8.2 Client-Specific Performance

The performance of individual client models is shown in Table 6.3. Notably, all clients exhibit higher MSE values compared to the global model, demonstrating the advantage of FL in achieving better generalization across diverse data distributions.

Table 6.3: Client-Specific Model Performance on CMAPSS Dataset

Client	**MSE**	**MAE**	**R^2 Score**
Client 1	1019.769	21.723	0.7916
Client 2	1215.665	24.389	0.7706
Client 3	634.825	18.520	0.8342
Client 4	2046.207	32.109	0.6339

6.8.3 Local vs Global Model Comparison

To further assess model effectiveness, Fig. 6.4 compares the Mean Squared Error (MSE) for local client models against the global model. The federated approach achieves lower MSE, demonstrating improved generalisation across heterogeneous clients.

Similarly, Fig. 6.5 shows the comparison of R^2 scores, where the global model consistently outperforms individual local models.

6.8.4 Classification Results-CMAPSS

Table 6.4 provides the classification metrics for the global model. The overall accuracy achieved is **70%**. The model performs well in detecting "low" class instances but shows moderate performance in distinguishing "medium" and "high" categories.

Table 6.4: Classification report for CMAPSS dataset

Class	**Precision**	**Recall**	**F1-Score**	**Support**
High	0.65	0.71	0.68	1200
Low	0.89	0.83	0.86	1600
Medium	0.58	0.52	0.55	1300
Weighted Avg	0.72	0.70	0.71	4100
Overall Accuracy	0.70			

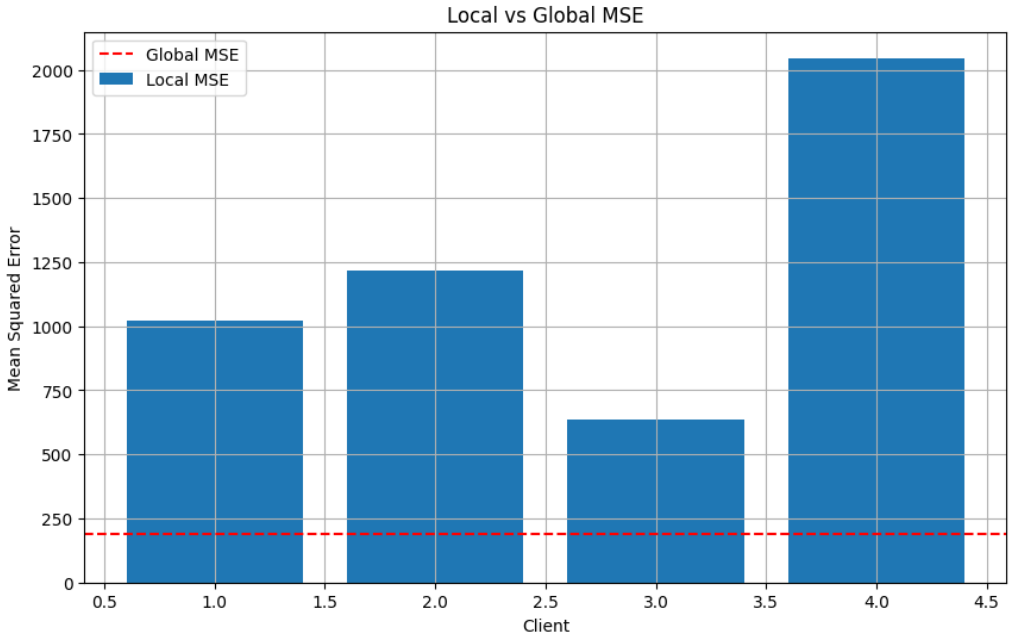

Figure 6.4: Local vs. Global MSE comparison-CMAPSS

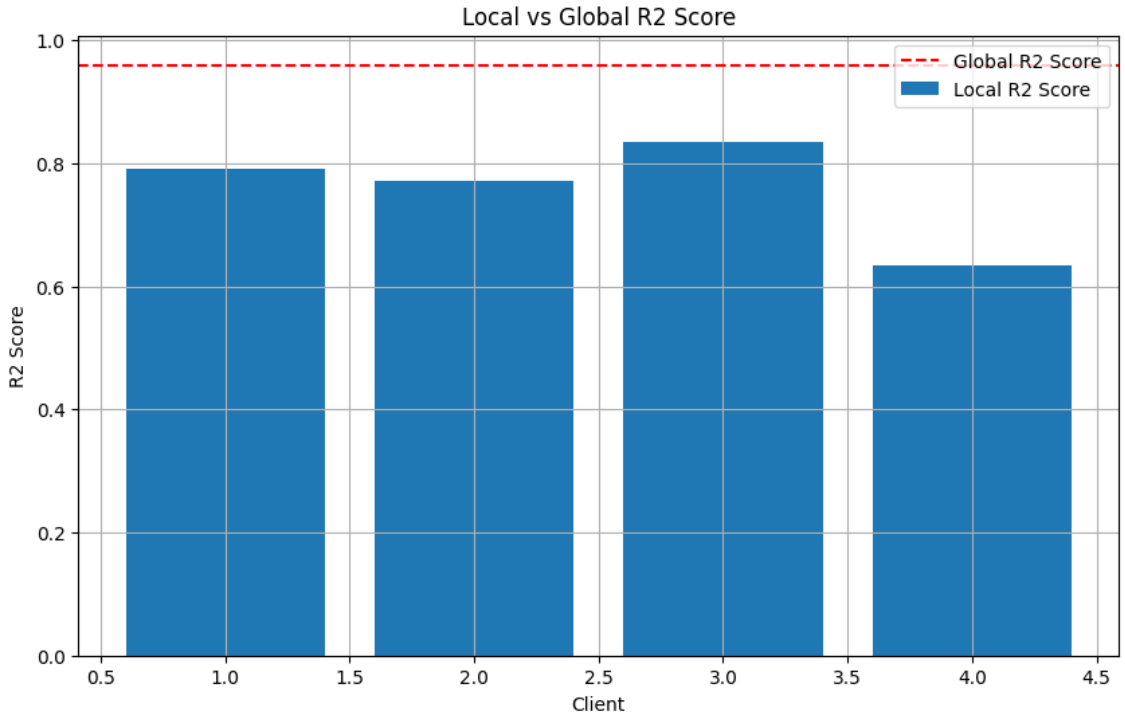

Figure 6.5: Local vs. Global R^2 score comparison-CMAPSS

The above CMAPSS results indicate that our proposed framework improves predictive performance in an FL setting. The global federated model outperforms individual clients, achieving a significantly lower MSE and a higher R^2 score. FedAdapt-CAD mitigates the effects of non-IID data, ensuring robust model performance across distributed clients. While the model achieves strong performance, classification for medium and high categories requires further optimization.

6.9 BATADAL Dataset Results

This dataset was used to evaluate the performance of our AD framework in water distribution systems. The results demonstrate the effectiveness of FL across multiple clients.

6.9.1 Global Model Performance

Table 6.5 presents the classification metrics for the global federated model. The model achieves 81% accuracy, with strong precision for the normal (0.0) class but poor detection of attack (1.0) instances.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0.0 (Normal)	0.81	1.00	0.89	1682
1.0 (Attack)	0.00	0.00	0.00	407
Overall Accuracy	0.81			
Macro Avg	0.40	0.50	0.45	2089
Weighted Avg	0.65	0.81	0.72	2089

Table 6.5: Global Model Classification Performance on BATADAL Dataset

6.9.2 Client-Specific Performance

The federated model was trained across five clients, each contributing data from different industrial regions. The accuracy for all client models was identical at 80.52%, as summarised in Table 6.6.

Table 6.6: Client Model Performance on BATADAL Dataset

Client	**Accuracy**
Client 1	0.8052
Client 2	0.8052
Client 3	0.8052
Client 4	0.8052
Client 5	0.8052

6.9.3 Challenges in Attack Detection

Although the global model achieves high accuracy, the recall for attack detection (1.0 class) is extremely low. This suggests that the dataset may be highly imbalanced, favouring normal operation states. The attack features may be difficult to distinguish without additional preprocessing. More advanced aggregation techniques may be needed to enhance attack classification. These results highlight the strengths and limitations of our framework in the context of BATADAL. High global model accuracy (81%) shows FL is effective for normal operation prediction. Consistent client performance indicates stable model training across distributed nodes. Poor recall for attack detection suggests improvements are needed in class balancing and feature extraction. Future work will explore adversarial training and data augmentation techniques to improve AD in federated settings.

6.10 WADI Dataset Results

The WADI dataset results demonstrate a near-perfect classification performance across clients and the global model.

6.10.1 Global Model Performance

Table 6.7 presents the classification report for the global model. The model achieves 99.67% accuracy, with precision, recall, and F1-score values near 100% across all classes.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
High	1.00	1.00	1.00	51290
Low	1.00	1.00	1.00	53820
Medium	1.00	0.99	1.00	51805
Overall Accuracy	0.9967			
Macro Avg	1.00	1.00	1.00	156915
Weighted Avg	1.00	1.00	1.00	156915

Table 6.7: Global Model Classification Performance on WADI Dataset

6.10.2 Client-Specific Performance

FL was conducted across five clients, and Table 6.8 summarizes their local model accuracy over five rounds of training.

6.10.3 Cross-Validation and FL Trends

Fig. 6.6 illustrates the cross-validation accuracy over different folds, confirming the model's generalization performance. Additionally, Fig. 6.7 presents the accuracy trends over five rounds for both local client models and the global federated model.

Round	Client 1	Client 2	Client 3	Client 4	Client 5
1	0.9953	0.9958	0.9949	0.9943	0.9948
2	0.9953	0.9958	0.9949	0.9943	0.9948
3	0.9953	0.9958	0.9949	0.9943	0.9948
4	0.9953	0.9958	0.9949	0.9943	0.9948
5	0.9953	0.9958	0.9949	0.9943	0.9948

Table 6.8: Client Model Performance on WADI Dataset

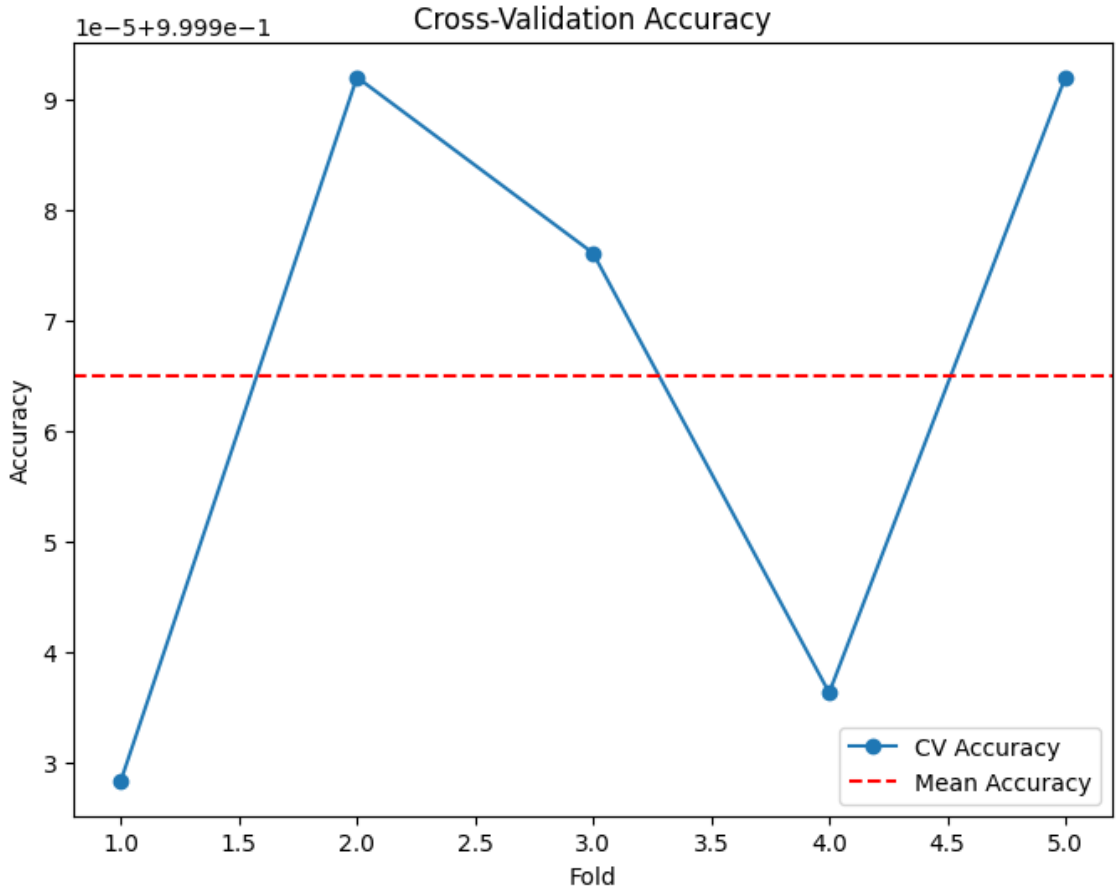

Figure 6.6: Cross-Validation accuracy on WADI dataset

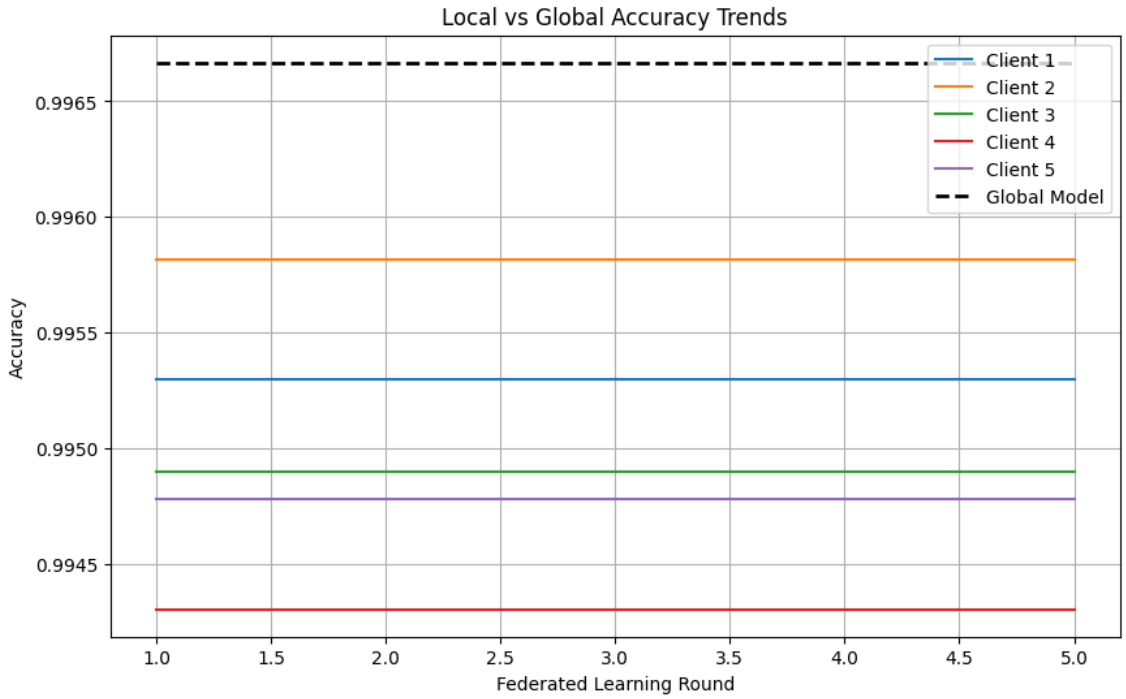

Figure 6.7: Local vs. Global accuracy trends (WADI dataset)

The global model consistently outperforms local models, demonstrating the advantage of federated aggregation.

Fig. 6.7 shows accuracy over five rounds, computed on each client's local test set. The global model is evaluated by averaging its performance across all clients. Close values reflect consistent performance and effective aggregation in non-IID settings.

These results highlight the effectiveness of the FedAdapt-CAD framework for WADI. Extremely high accuracy (99.67%) confirms the suitability of FL for industrial AD. Stable local model accuracy across five rounds demonstrates consistent client training performance. Thus, FedAdapt-CAD enhances generalisation, as observed in the comparison between local and global models. Minimal misclassification is observed, making this approach highly reliable for real-world deployment.

6.11 Compared Federated Learning Efficiency Analysis Across Datasets

- WADI and BATADAL exhibited stable accuracy trends across rounds, whereas CMAPSS showed higher variance due to non-IID effects.
- Federated aggregation significantly improved model performance in CMAPSS and BATADAL, but its impact was less pronounced in WADI due to the already high baseline accuracy.

Despite achieving high performance, some limitations remain: **Class Imbalance in BATADAL**: Future enhancements could involve federated cost-sensitive learning, where minority classes receive higher weight in model aggregation. **Optimization for Non-IID Data**: CMAPSS demonstrated the challenges of heterogeneous client

data, which could be addressed using adaptive federated averaging techniques. **Communication Efficiency**: While WADI achieved near-perfect accuracy, reducing model update overhead for large-scale federated deployments remains a key area for improvement.

6.11.1 Performance Comparison with State-of-the-Art

Compared to existing FL models from the literature, such as FedH2L [169], HFL-EdgeCloud [167], and SecureFed [194], FedAdapt-CAD offers better adaptability via CAA and DMA, yielding superior precision, recall, and convergence across domains. FedAdapt-CAD consistently enhances model generalization in non-IID federated settings, outperforming client-specific models across industrial domains. The evaluation on the WADI dataset highlights its capability to achieve near-perfect anomaly classification, making it highly suitable for deployment in real-time industrial environments where accuracy and stability are critical. However, in the BATADAL dataset, the model's performance was hindered by class imbalance, especially in detecting rare attack classes. This underscores the importance of future enhancements, such as cost-sensitive FL or adversarial training, to improve robustness. Below is the comparison table that validates our approach results with others.

Table 6.9: Performance Comparison of FL Methods Across WADI, CMAPSS, and BATADAL (Same Experimental Settings)

Model	**WADI**			**CMAPSS**			**BATADAL**		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
FedAvg	96.12%	0.94	0.92	60.14%	0.60	0.61	74.63%	0.36	0.41
FedProx	97.25%	0.96	0.95	64.22%	0.64	0.65	77.24%	0.38	0.46
FedH2L	98.03%	0.98	0.97	66.85%	0.67	0.66	78.90%	0.39	0.48
FedAdapt-CAD (Ours)	**99.67%**	**1.00**	**1.00**	**68.26%**	**0.69**	**0.69**	**81.00%**	**0.40**	**0.50**

6.12 Conclusion

The FedAdapt-CAD framework is a robust and scalable approach for AD, predictive maintenance, and cyber-attack detection across diverse industrial environments. It integrates CAA and DMA to address challenges such as non-IID data distribution, model performance variability, and real-time adaptation to evolving threats. Experimental evaluations on the BATADAL, CMAPSS, and WADI datasets show that our framework significantly improves classification accuracy and fairness, particularly in complex, heterogeneous environments. However, it still faces challenges in communication efficiency and in handling highly imbalanced datasets, which may affect detection rates for rare cyber-attacks. Future work will focus on enhancing model optimisation strategies, reducing communication overhead, and improving attack detection through adversarial training techniques and federated cost-sensitive learning. The framework's security and scalability will be enhanced by incorporating privacy-preserving mechanisms like secure multi-party computation (SMPC) and differential privacy.

Chapter 7

Conclusion and Future Work

This dissertation initially examined the importance of utilising unsupervised ML techniques in industrial settings. The dataset of Marposs S.p.A. DT machines was analysed using SPC analysis and Python's visualisation libraries (seaborn, matplotlib, and pyplot). Data normalisation, feature extraction, and ARIMA and SARIMA analyses were performed to assess seasonality and patterns in the industrial dataset. Dataset grouping was performed using our custom algorithm to prepare the data for training. SOTA unsupervised ML algorithms (K-Means, DBSCAN, and Isolation Forest) were implemented to identify anomalous behavior in manufacturing processes. The complete procedure and its experimental validation have been discussed in detail in Chapter 3. The implemented algorithms helped predict anomalies in industrial datasets.

The second significant contribution of this dissertation was the exploration of trending AI technologies and best practices of MLOps that can be used in industrial DTs for automation and AD. We proposed an industrial DT framework, C2E-AIOps, which was designed and implemented for our industrial collaboration partner, Marposs S.p.A. The framework is flexible and can be adapted to many industrial use cases. We converted our Proof-of-Concept into a production-ready implementation. In Chapter 4, we have discussed the C2E-AIOps framework and its workflow in detail. This is an IEC-based framework that exploited Azure IoT Central, Microsoft Azure, and its AI-based services, Azure ML/AI Studio, and AutoML to train ML models on industrial datasets. The ONNX standard is used to deploy those models on industrial edge devices. We containerized the ML model inference using Docker technology. At the edge layer, GitHub Actions was used for CI/CD implementation of our model's containerized solution for our industrial use case. We also leveraged Azure Function, a serverless application of MS Azure that automates the execution of our C2E-AIOps framework in our industrial use case. In Chapter 5, we discussed the system requirements to implement the C2E-AIOps in an industrial setting. Moreover, in-the-field experimental results demonstrate the effectiveness of our framework. The working codes have been built in Python and made available in the GitHub repository [145, 146, 150].

The third significant contribution of this dissertation is the introduction of FedAdapt-CAD, an innovative FL framework designed to preserve the privacy of industrial datasets and threat detection in industrial control systems. Chapter 6 discussed the framework, its implementation details, and experimental results achieved. FedAdapt-CAD introduces two novel components: *CAA* and *DMA*. *CAA*

is a dynamic aggregation strategy at the client level that improves convergence and fairness in non-IID settings. And the DMA mechanism adapts to changes in data distribution and cyber attack patterns at the server level. The framework is rigorously evaluated on three real-world industrial datasets, WADI, CMAPSS, and BATADAL, which cover a range of industrial AD scenarios. Experimental evaluations on BATADAL, CMAPSS, and WADI datasets show that the proposed framework significantly enhances classification accuracy and fairness, particularly in complex and heterogeneous environments. However, it still faces challenges in communication efficiency and handling highly imbalanced datasets.

The limitation of C2E-AIOps framework is that it is limited to the Microsoft Azure cloud. Other cloud technologies, such as Amazon Web Services (AWS) and Google Cloud, could be leveraged to make the solution more generic.

Currently, C2E-AIOps is implemented for a specific industrial use case. Moreover, it can be implemented on other industrial use cases. To make our C2E-AIOps more adaptable in diverse manufacturing settings, we can test other industrial-grade protocols such as *Modbus*, *OPC-UA*, or *Profinet*. Currently, we have deployed the Azure Function in our local industrial system. In the future, Azure Function deployment will be enhanced by incorporating it as a complete, distributed cloud-based solution, deploying it as a Function App in Microsoft Azure. Also, we plan to make our C2E-AIOps an Infrastructure as Code (IAC) service by leveraging Azure Logic Apps [195]. This platform significantly reduces the need for coding when integrating workflows with resources from multiple components, including services, systems, applications, and data sources. In the context of FedAdapt-CAD, future work will focus on enhancing model optimisation strategies, reducing communication overhead, and improving attack detection through adversarial training techniques and federated cost-sensitive learning. The framework's scalability will be enhanced by incorporating privacy-preserving mechanisms, such as secure multi-party computation (SMPC) and differential privacy. Additionally, to improve efficiency and speed up activities on a large scale, the C2E-AIOps framework with FL contributions can be implemented using a 5G/6G network.

BIBLIOGRAPHY

[1] J. Lee, H.-A. Kao, and S. Yang, "Service innovation and smart analytics for industry 4.0 and big data environment," *Procedia cirp*, vol. 16, pp. 3–8, 2014.

[2] L. Kaupp, H. Webert, K. Nazemi, B. Humm, and S. Simons, "Context: An industry 4.0 dataset of contextual faults in a smart factory," *Procedia Computer Science*, vol. 180, pp. 492–501, 2021.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[4] A. Chatterjee and B. S. Ahmed, "Iot anomaly detection methods and applications: A survey," *Internet of Things*, vol. 19, p. 100568, 2022.

[5] F. Huch, M. Golagha, A. Petrovska, and A. Krauss, "Machine learning-based run-time anomaly detection in software systems: An industrial evaluation," in *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2018, pp. 13–18.

[6] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless personal communications*, vol. 58, no. 1, pp. 49–69, 2011.

[7] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of industrial information integration*, vol. 6, pp. 1–10, 2017.

[8] J. Cunha, N. Batista, C. Cardeira, and R. Melicio, "Upgrading a legacy manufacturing cell to iot," *Journal of Sensor and Actuator Networks*, vol. 10, no. 4, p. 65, 2021.

[9] S. Ayvaz and K. Alpay, "Predictive maintenance system for production lines in manufacturing: A machine learning approach using iot data in real-time," *Expert Systems with Applications*, vol. 173, p. 114598, 2021.

[10] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[11] A. Angelopoulos, E. T. Michailidis, N. Nomikos, P. Trakadas, A. Hatziefremidis, S. Voliotis, and T. Zahariadis, "Tackling faults in the industry 4.0 era—a survey of machine-learning solutions and key aspects," *Sensors*, vol. 20, no. 1, p. 109, 2019.

[12] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Production & manufacturing research*, vol. 4, no. 1, pp. 23–45, 2016.

[13] S. F. Chevtchenko, E. D. S. Rocha, M. C. M. Dos Santos, R. L. Mota, D. M. Vieira, E. C. De Andrade, and D. R. B. De Araújo, "Anomaly detection in industrial machinery using iot devices and machine learning: a systematic mapping," *IEEE Access*, vol. 11, pp. 128 288–128 305, 2023.

[14] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *Ifac-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018.

[15] R. Rosen, G. Von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *Ifac-papersonline*, vol. 48, no. 3, pp. 567–572, 2015.

[16] G. Schuh, R. Anderl, J. Gausemeier, M. ten Hompel, and W. Wahlster, "Industrie 4.0 maturity index," *Managing the digital transformation of companies*, vol. 61, 2017.

[17] B. Schleich, N. Anwer, L. Mathieu, and S. Wartzack, "Shaping the digital twin for design and production engineering," *CIRP annals*, vol. 66, no. 1, pp. 141–144, 2017.

[18] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing letters*, vol. 3, pp. 18–23, 2015.

[19] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, pp. 3563–3576, 2018.

[20] F. Tao, M. Zhang, and A. Y. C. Nee, *Digital twin driven smart manufacturing*. Academic press, 2019.

[21] G. Amirkhanova, B. Amirkhanov, G. Tyulepberdinova, and T. Ishmurzin, "Application of machine learning algorithms in digital twin monitoring systems: An overview of approaches, methods, and prospects," in *2024 International Conference on Intelligent Computing and Next Generation Networks (ICNGN)*. IEEE, 2024, pp. 01–05.

[22] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *Journal of manufacturing systems*, vol. 48, pp. 157–169, 2018.

[23] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical systems and signal processing*, vol. 20, no. 7, pp. 1483–1510, 2006.

[24] M. Fahim and A. Sillitti, "Anomaly detection, analysis and prediction techniques in iot environment: A systematic literature review," *IEEE Access*, vol. 7, pp. 81 664–81 681, 2019.

[25] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE transactions on knowledge and data engineering*, vol. 24, no. 5, pp. 823–839, 2010.

[26] Y. Wang, J. Fang, Y. Cheng, H. She, Y. Guo, and G. Zheng, "Cooperative end-edge-cloud computing and resource allocation for digital twin enabled 6g industrial iot," *IEEE Journal of Selected Topics in Signal Processing*, vol. 18, no. 1, pp. 124–137, 2023.

[27] A. Al-Dulaimy, M. Jansen, B. Johansson, A. Trivedi, A. Iosup, M. Ashjaei, A. Galletta, D. Kimovski, R. Prodan, K. Tserpes *et al.*, "The computing continuum: From iot to the cloud," *Internet of Things*, vol. 27, p. 101272, 2024.

[28] E. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and us air force vehicles," in *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, 2012, p. 1818.

[29] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia manufacturing*, vol. 11, pp. 939–948, 2017.

[30] D. Bowman, L. Dwyer, A. Levers, E. A. Patterson, S. Purdie, and K. Vikhorev, "A unified approach to digital twin architecture—proof-of-concept activity in the nuclear sector," *Ieee Access*, vol. 10, pp. 44 691–44 709, 2022.

[31] A. Kychkin and A. Nikolaev, "Iot-based mine ventilation control system architecture with digital twin," in *2020 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*. IEEE, 2020, pp. 1–5.

[32] T. Lyu, U. D. Atmojo, and V. Vyatkin, "Towards cloud-based virtual commissioning of distributed automation applications with iec 61499 and containerization technology," in *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2021, pp. 1–7.

[33] C. Liu, P. Jiang, and W. Jiang, "Web-based digital twin modeling and remote control of cyber-physical production systems," *Robotics and computer-integrated manufacturing*, vol. 64, p. 101956, 2020.

[34] G. Shao, S. Jain, C. Laroque, L. H. Lee, P. Lendermann, and O. Rose, "Digital twin for smart manufacturing: the simulation aspect," in *2019 Winter Simulation Conference (WSC)*. IEEE, 2019, pp. 2085–2098.

[35] Markets and Markets, "Digital twin market," https://www.marketsandmarkets.com/Market-Reports/digital-twin-market-225269522.html, 2023, accessed: 2025-07-01.

[36] P. Bellavista and G. Di Modica, "Iotwins: implementing distributed and hybrid digital twins in industrial manufacturing and facility management settings," *Future Internet*, vol. 16, no. 2, p. 65, 2024.

[37] I. A. Bartsiokas, P. K. Gkonis, D. I. Kaklamani, and I. S. Venieris, "Ml-based radio resource management in 5g and beyond networks: A survey," *IEEE Access*, vol. 10, pp. 83 507–83 528, 2022.

[38] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D'Andria, "A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues," *Future Internet*, vol. 15, no. 12, p. 383, 2023.

[39] S. Hu, X. Chen, W. Ni, E. Hossain, and X. Wang, "Distributed machine learning for wireless communication networks: Techniques, architectures, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1458–1493, 2021.

[40] F. Marozzo, A. Orsino, D. Talia, and P. Trunfio, "Edge computing solutions for distributed machine learning," in *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2022, pp. 1–8.

[41] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[42] K. J. Rahman, F. Ahmed, N. Akhter, M. Hasan, R. Amin, K. E. Aziz, A. M. Islam, M. S. H. Mukta, and A. N. Islam, "Challenges, applications and design aspects of federated learning: A survey," *IEEe Access*, vol. 9, pp. 124 682–124 700, 2021.

[43] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, "A survey of federated learning for edge computing: Research problems and solutions," *High-Confidence Computing*, vol. 1, no. 1, p. 100008, 2021.

[44] Marposs S.p.A., https://www.marposs.com.

[45] Microsoft Learn, "What is azure functions?" Mar. 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview

[46] M. Zinkevich, "Rules of machine learning: Best practices for ml engineering," *URL: https://developers. google. com/machine-learning/guides/rules-of-ml*, 2017.

[47] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[48] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," *ACM computing surveys*, vol. 55, no. 6, pp. 1–29, 2022.

[49] B. Chebel-Morello, S. Malinowski, and H. Senoussi, "Feature selection for fault detection systems: application to the tennessee eastman process," *Applied Intelligence*, vol. 44, no. 1, pp. 111–122, 2016.

[50] X. Gao and J. Hou, "An improved svm integrated gs-pca fault diagnosis approach of tennessee eastman process," *Neurocomputing*, vol. 174, pp. 906–911, 2016.

[51] M. Adeli and A. Mazinan, "High efficiency fault-detection and fault-tolerant control approach in tennessee eastman process via fuzzy-based neural network representation," *Complex & Intelligent Systems*, vol. 6, no. 1, pp. 199–212, 2020.

[52] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot," *Journal of network and computer applications*, vol. 149, p. 102481, 2020.

[53] F. de Assis Boldt, T. W. Rauber, and F. M. Varejão, "Cascade feature selection and elm for automatic fault diagnosis of the tennessee eastman process," *Neurocomputing*, vol. 239, pp. 238–248, 2017.

[54] R. Marino, C. Wisultschew, A. Otero, J. M. Lanza-Gutierrez, J. Portilla, and E. de la Torre, "A machine-learning-based distributed system for fault diagnosis with scalable detection quality in industrial iot," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4339–4352, 2020.

[55] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.

[56] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of process control*, vol. 22, no. 9, pp. 1567–1581, 2012.

[57] M. Albayati, J. Faraj, A. Thompson, P. Patil, R. Gorthala, and S. Rajasekaran, "Semi-supervised machine learning for fault detection and diagnosis of a rooftop unit. big data min. anal. 6 (2), 170–184 (2023)," 2022.

[58] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[59] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE access*, vol. 7, pp. 65 579–65 615, 2019.

[60] H. Bangui, M. Ge, and B. Buhnova, "Exploring big data clustering algorithms for internet of things applications." in *IoTBDS*, 2018, pp. 269–276.

[61] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *expert systems with applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[62] Y. Wang, S.-T. Xia, Q. Tang, J. Wu, and X. Zhu, "A novel consistent random forest framework: Bernoulli random forests," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3510–3523, 2018.

[63] H. Frigui, "Unsupervised learning of arbitrarily shaped clusters using ensembles of gaussian models," *Pattern analysis and applications*, vol. 8, no. 1, pp. 32–49, 2005.

[64] Z. Qi, Y. Tian, Y. Shi, and Y. Shi, "An effective and efficient hierarchical k-means clustering algorithm," *International Journal of Distributed Sensor Networks*, vol. 13, no. 8, p. 1550147717728627, 2017.

[65] D. Velasquez, E. Perez, X. Oregui, A. Artetxe, J. Manteca, J. E. Mansilla, M. Toro, M. Maiza, and B. Sierra, "A hybrid machine-learning ensemble for anomaly detection in real-time industry 4.0 systems," *IEEE Access*, vol. 10, pp. 72 024–72 036, 2022.

[66] M. Bahri, F. Salutari, A. Putina, and M. Sozio, "Automl: state of the art with a focus on anomaly detection, challenges, and research directions," *International Journal of Data Science and Analytics*, vol. 14, no. 2, pp. 113–126, 2022.

[67] "Autoweka – automl for tabular data," https://www.automl.org/automl-for-x/tabular-data/autoweka/, AutoML.org, 2025.

[68] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *Journal of Machine Learning Research*, vol. 18, no. 25, pp. 1–5, 2017.

[69] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 847–855.

[70] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[71] AutoML.org, "Auto-sklearn – automl for tabular data," AutoML.org, 2025. [Online]. Available: https://www.automl.org/automl-for-x/tabular-data/auto-sklearn/

[72] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in neural information processing systems*, vol. 28, 2015.

[73] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn," in *Proceedings of the ICML 2014 AutoML Workshop*, 2014. [Online]. Available: https://compneuro.uwaterloo.ca/files/publications/komer.2014b.pdf

[74] J. Bergstra, D. Yamins, D. D. Cox *et al.*, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms." *SciPy*, vol. 13, p. 20, 2013.

[75] N. Iam-On and T. Boongoen, "Comparative study of matrix refinement approaches for ensemble clustering," *Machine Learning*, vol. 98, no. 1, pp. 269–300, 2015.

[76] Y. Jiang and N. Verma, "Meta-learning to cluster," *arXiv preprint arXiv:1910.14134*, 2019.

[77] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.

[78] M. Carnein, H. Trautmann, A. Bifet, and B. Pfahringer, "confstream: Automated algorithm selection and configuration of stream clustering algorithms," in *International conference on learning and intelligent optimization*. Springer, 2020, pp. 80–95.

[79] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren, "The online performance estimation framework: heterogeneous ensemble learning for data streams," *Machine Learning*, vol. 107, no. 1, pp. 149–176, 2018.

[80] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of machine learning research*, vol. 20, no. 96, pp. 1–7, 2019.

[81] Y. Li, D. Zha, P. Venugopal, N. Zou, and X. Hu, "Pyodds: An end-to-end outlier detection system with automated machine learning," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 153–157.

[82] Y. Zhao, Z. Nasrullah, M. K. Hryniewicki, and Z. Li, "Lscp: Locally selective combination in parallel outlier ensembles," in *Proceedings of the 2019 SIAM international conference on data mining*. SIAM, 2019, pp. 585–593.

[83] K.-H. Lai, D. Zha, G. Wang, J. Xu, Y. Zhao, D. Kumar, Y. Chen, P. Zumkhawaka, M. Wan, D. Martinez *et al.*, "Tods: An automated time series outlier detection system," in *Proceedings of the aaai conference on artificial intelligence*, vol. 35, no. 18, 2021, pp. 16 060–16 062.

[84] Y. Zhao, R. A. Rossi, and L. Akoglu, "Automating outlier detection via meta-learning," *arXiv preprint arXiv:2009.10606*, 2020.

[85] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac–instance-specific algorithm configuration," in *ECAI 2010*. Ios Press, 2010, pp. 751–756.

[86] E. Burnaev, P. Erofeev, and D. Smolyakov, "Model selection for anomaly detection," in *Eighth International Conference on Machine Vision (ICMV 2015)*, vol. 9875. SPIE, 2015, pp. 445–450.

[87] Y. Xiao, H. Wang, L. Zhang, and W. Xu, "Two methods of selecting gaussian kernel parameters for one-class svm and their application to fault detection," *Knowledge-Based Systems*, vol. 59, pp. 75–84, 2014.

[88] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012.

[89] A. Ghoting, S. Parthasarathy, and M. E. Otey, "Fast mining of distance-based outliers in high-dimensional datasets," *Data Mining and Knowledge Discovery*, vol. 16, no. 3, pp. 349–364, 2008.

[90] E. Raj, D. Buffoni, M. Westerlund, and K. Ahola, "Edge mlops: An automation framework for aiot applications," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 191–200.

[91] K. Sato, "What is ml ops? best practices for devops ml," *Presentation at Google Cloud Next'18*, 2018.

[92] D. A. Tamburri, "Sustainable mlops: Trends and challenges," in *2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC)*. IEEE, 2020, pp. 17–23.

[93] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[94] Y. Lee, A. Scolari, B.-G. Chun, M. Weimer, and M. Interlandi, "From the edge to the cloud: Model serving in ml. net." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 46–53, 2018.

[95] P. Agrawal and N. Rawat, "Devops, a new approach to cloud development & testing," in *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, vol. 1. IEEE, 2019, pp. 1–4.

[96] J. Ereth, "Dataops-towards a definition." *LWDA*, vol. 2191, no. 104-112, p. 106, 2018.

[97] A. R. Munappy, D. I. Mattos, J. Bosch, H. H. Olsson, and A. Dakkak, "From ad-hoc data analytics to dataops," in *Proceedings of the International Conference on Software and System Processes*, 2020, pp. 165–174.

[98] F. Tao and M. Zhang, "Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing," *IEEE access*, vol. 5, pp. 20 418–20 427, 2017.

[99] Y. Lu, C. Liu, I. Kevin, K. Wang, H. Huang, and X. Xu, "Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robotics and computer-integrated manufacturing*, vol. 61, p. 101837, 2020.

[100] V. Damjanovic-Behrendt and W. Behrendt, "An open source approach to the design and implementation of digital twins for smart manufacturing," *International Journal of Computer Integrated Manufacturing*, vol. 32, no. 4-5, pp. 366–384, 2019.

[101] C. Zhuang, J. Liu, and H. Xiong, "Digital twin-based smart production management and control framework for the complex product assembly shop-floor," *The international journal of advanced manufacturing technology*, vol. 96, no. 1, pp. 1149–1163, 2018.

[102] J. Leng, Q. Liu, S. Ye, J. Jing, Y. Wang, C. Zhang, D. Zhang, and X. Chen, "Digital twin-driven rapid reconfiguration of the automated manufacturing system via an open architecture model," *Robotics and computer-integrated manufacturing*, vol. 63, p. 101895, 2020.

[103] Z. Jiang, Y. Guo, and Z. Wang, "Digital twin to improve the virtual-real integration of industrial iot," *Journal of Industrial Information Integration*, vol. 22, p. 100196, 2021.

[104] M.-H. Hung, Y.-C. Lin, H.-C. Hsiao, C.-C. Chen, K.-C. Lai, Y.-M. Hsieh, H. Tieng, T.-H. Tsai, H.-C. Huang, H.-C. Yang *et al.*, "A novel implementation framework of digital twins for intelligent manufacturing based on container technology and cloud manufacturing services," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1614–1630, 2022.

[105] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices," in *Proceedings of the 10th International Conference on the Internet of Things*, 2020, pp. 1–8.

[106] J. Moon, S. Kum, and S. Lee, "A heterogeneous iot data analysis framework with collaboration of edge-cloud computing: Focusing on indoor pm10 and pm2. 5 status prediction," *Sensors*, vol. 19, no. 14, p. 3038, 2019.

[107] A. Bhattacharjee, Y. Barve, S. Khare, S. Bao, Z. Kang, A. Gokhale, and T. Damiano, "Stratum: A bigdata-as-a-service for lifecycle management of iot analytics applications," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1607–1612.

[108] Siemens AG, "Artis – genior modular tool and process monitoring system," https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10166107, Jul. 2025.

[109] Marposs S.p.A., "Data base management system," https://www.marposs.com/eng/product/data-base-management-system, 2025.

[110] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay, *scikit-learn: Machine Learning in Python*, https://scikit-learn.org/stable/index.html, scikit-learn developers, 2024, accessed: 2025-07-26.

[111] ——, *scikit-learn: Machine Learning in Python*, https://scikit-learn.org/, 2011, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[112] D. ElMenshawy, W. Helmy, and N. El-Tazi, "A clustering based approach for contextual anomaly detection in internet of things," *J. Comput. Sci*, vol. 15, no. 8, pp. 1195–1202, 2019.

[113] scikit-learn developers, *sklearn.cluster.DBSCAN — scikit-learn documentation*, https://scikit-learn.org/, scikit-learn, 2024, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[114] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[115] A. Lavin and S. Ahmad, "Numenta anomaly benchmark (nab)," https://github.com/numenta/NAB, 2015, accessed: 2025-07-26.

[116] scikit-learn developers, *Clustering performance evaluation — scikit-learn documentation*, https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation, scikit-learn, 2024, accessed: 2025-07-26.

[117] Microsoft Corporation, "Azure machine learning," https://azure.microsoft.com/en-us/products/machine-learning, 2025. [Online]. Available: https://azure.microsoft.com/enâĂŚus/products/machineâĂŚlearning

[118] ——, "What is azure iot hub?" https://learn.microsoft.com/en-us/azure/iot-hub/about-iot-hub, 2025, accessed: 2025-07-28. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-hub/about-iot-hub

[119] ——, "Azure machine learning studio," https://learn.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning?view=azureml-api-2, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning?view=azureml-api-2

[120] ——, "What is automated machine learning (automl)?" https://learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml

[121] ——, "Azure blob storage," https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview, 2025, accessed: 2025-07-28. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview

[122] ——, "Azure IoT Central," 2024. [Online]. Available: https://azure.microsoft.com/en-us/products/iot-central

[123] GitHub, Inc., "Github: Where the world builds software," https://github.com, 2025. [Online]. Available: https://github.com

[124] ——, "Github actions documentation," https://docs.github.com/en/actions, 2025. [Online]. Available: https://docs.github.com/en/actions

[125] Microsoft Corporation, "What is azure iot edge?" https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge

[126] Docker, Inc., "Docker: Empowering app development for developers," https://www.docker.com, 2025. [Online]. Available: https://www.docker.com

[127] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data,"

in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 54. PMLR, 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf

[128] O. C. M. . Facebook), "Open neural network exchange (onnx)," https://onnx.ai/, 2017, open-source format for interoperable machine learning models.

[129] ONNX Community, "onnxmltools: Convert ML models to ONNX," https://github.com/onnx/onnxmltools, 2025.

[130] Microsoft Corporation, "Onnx runtime," https://onnxruntime.ai/, 2024.

[131] Apple Inc., "coremltools," https://github.com/apple/coremltools, 2025.

[132] ONNX Runtime developers, "Getting started: Converting tensorflow to onnx," https://onnxruntime.ai/docs/tutorials/tf-get-started.html, 2025, accessed: 2025-07-30.

[133] Eclipse Mosquitto Project, "Eclipse Mosquitto – an open source mqtt broker," https://mosquitto.org/, 2025.

[134] Y. Zhou, Y. Yu, and B. Ding, "Towards mlops: A case study of ml pipeline platform," in *2020 International conference on artificial intelligence and computer engineering (ICAICE)*. IEEE, 2020, pp. 494–500.

[135] S. Mäkinen, H. Skogström, E. Laaksonen, and T. Mikkonen, "Who needs mlops: What data scientists seek to accomplish and how can mlops help?" in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 109–112.

[136] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," *IEEE access*, vol. 11, pp. 31 866–31 879, 2023.

[137] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg, and S. Ahlawat, "On continuous integration/continuous delivery for automated deployment of machine learning models using mlops," in *2021 IEEE fourth international conference on artificial intelligence and knowledge engineering (AIKE)*. IEEE, 2021, pp. 25–28.

[138] R. Dave, "Implementing mlops on edge-cloud systems: A new paradigm for training at the edge," 2023.

[139] MLflow Developers, "MLflow - a platform for the machine learning lifecycle," https://mlflow.org/, 2025.

[140] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[141] Kubeflow Project Contributors, "Kubeflow: The machine learning toolkit for kubernetes," https://www.kubeflow.org/, 2025.

[142] TensorFlow Team, "Tensorflow extended (tfx): A production-ready machine learning platform," https://www.tensorflow.org/tfx, 2025.

[143] I. Kumara, F. Pecorelli, G. Catolino, R. Kazman, D. A. Tamburri, and W.-J. Van Den Heuvel, "Architecting mlops in the cloud: From theory to practice," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2023, pp. 333–335.

[144] Marposs S.p.A., "MAINDO: Marposs Digital Platform," https://www.marposs.com/eng/marposs-digital-platform-maindo/, 2023.

[145] A. Farooq, "ONNX-CI-CD-Pipeline: Automated ci/cd setup for onnx," https://github.com/Azaz-farooq/ONNX-CI-CD-Pipeline, 2025.

[146] Azaz-Farooq, "Automlpipeline," https://github.com/Azaz-farooq/AutoMLPipeline, 2025.

[147] Project Jupyter, "Project jupyter — open source data science software," https://jupyter.org/.

[148] EMQ Technologies Inc., "MQTTX: A Powerful MQTT Client Toolbox (Desktop, CLI & Web)," https://mqttx.app/, 2025, version 1.12.0 available as of mid-2025. [Online]. Available: https://mqttx.app/

[149] Microsoft Azure, "Azure functions pricing," https://azure.microsoft.com/en-us/pricing/details/functions/, 2025.

[150] Azaz-Farooq, "Azurefunction," https://github.com/Azaz-farooq/AzureFunction, 2025.

[151] Microsoft Learn, "Azure functions http triggers and bindings," 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger?tabs=in-process%2Cfunctionsv2&pivots=programming-language-csharp

[152] Microsoft Corporation, "Microsoft Azure Pricing Calculator," https://azure.microsoft.com/en-us/pricing/calculator/, 2023.

[153] GitHub, "About billing for GitHub Actions," GitHub, Inc. [Online]. Available: https://docs.github.com/en/billing/concepts/product-billing/github-actions

[154] J. B. Blychert and J. Kilit, "Edge computing and gdpr: A technical security and legal compliance analysis," Master's Thesis, School of Engineering, Jönköping University, May 2025, final thesis carried out at the School of Engineering, authors responsible for opinions and conclusions.

[155] Wikipedia contributors, "Iso/IEC 27701," https://en.wikipedia.org/wiki/ISO/IEC_27701.

[156] L. Albshaier, S. Almarri, and A. Albuali, "Federated learning for cloud and edge security: A systematic review of challenges and ai opportunities," *Electronics*, vol. 14, no. 5, p. 1019, 2025.

[157] P. Bellavista, L. Foschini, and A. Mora, "Decentralised learning in federated deployment environments: A system-level survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–38, 2021.

[158] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[159] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[160] "GDPR.EU – Complete guide to GDPR compliance," https://gdpr.eu/, Proton Technologies AG, 2025, operated by Proton Technologies AG, co-funded by Horizon 2020 Framework Programme of the European Union; © 2025 Proton AG. [Online]. Available: https://gdpr.eu/

[161] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*. PMLR, 2020, pp. 5132–5143.

[162] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[163] Z. Li, Y. He, H. Yu, J. Kang, X. Li, Z. Xu, and D. Niyato, "Data heterogeneity-robust federated learning via group client selection in industrial iot," *arXiv preprint arXiv:2202.01512*, 2022.

[164] Z. Li, S. Zeng, H. Yu, Z. Zhang, L. Luo, and D. Niyato, "Vertical federated learning across heterogeneous regions for industry 4.0," *IEEE Transactions on Industrial Informatics*, 2023.

[165] B. Liu, N. Lv, Y. Guo, and Y. Li, "Recent advances on federated learning: A systematic survey," *arXiv preprint arXiv:2301.01299*, 2023.

[166] W. Fang, D.-J. Han, E. Chen, S. Wang, and C. G. Brinton, "Hierarchical federated learning with multi-timescale gradient correction," *arXiv preprint arXiv:2409.18448*, 2024.

[167] Y. Liu, P. Zhang, T. Chen, M. Hong, and Q. Yang, "Client-edge-cloud hierarchical federated learning," *arXiv preprint arXiv:2006.01088*, 2020.

[168] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Device-edge-cloud hierarchical federated learning," *arXiv preprint arXiv:2102.00046*, 2021.

[169] G. Li, Y. Hu, M. Zhang, J. Liu, and Q. Yin, "Fedh2l: Federated learning with hierarchical heterogeneous learning in industrial iot," *Journal of IoT Engineering*, vol. 5, no. 3, pp. 120–136, 2021.

[170] iTrust Lab, "itrust labs datasets," 2025.

[171] NASA, "C-MAPSS Jet Engine Simulated Data," 2025.

[172] BATADAL Competition Organizers, "BATADAL: Battle of the Attack Detection Algorithms Dataset," 2025.

[173] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[174] X. Yao, T. Huang, R.-X. Zhang, R. Li, and L. Sun, "Federated learning with unbiased gradient aggregation and controllable meta updating," *arXiv preprint arXiv:1910.08234*, 2019.

[175] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data," in *ICC 2020-2020 IEEE International Conference On Communications (ICC)*. IEEE, 2020, pp. 1–7.

[176] S. Dutta, I. L. de Freitas, P. M. Xavier, C. M. de Farias, and D. E. B. Neira, "Federated learning in chemical engineering: A tutorial on a framework for privacy-preserving collaboration across distributed data sources," *arXiv preprint arXiv:2411.16737*, 2024.

[177] L. Huang, J. Liu, Y. Zhang, and D. O. Tao, "Hierarchically fair federated learning," *arXiv preprint arXiv:2004.10386*, 2020.

[178] X. Xu, X. Yu, and L. Luo, "Hierarchical federated learning in smart manufacturing systems: Architecture and framework," *IEEE Transactions on Systems, Man, and Cybernetics*, 2022.

[179] R. Wang, T. Chen, and Y. Zhang, "Collaborative intelligence in federated learning for industrial iot," *IEEE IoT Journal*, 2022.

[180] C. Guo, M. Liu, and W. Zhang, "Edge-focused federated learning for manufacturing systems," *Journal of Manufacturing Systems*, vol. 65, pp. 145–160, 2023.

[181] X. Lin, M. Zhang, and Q. Zhao, "Adaptive federated learning with clustering in industrial iot networks," *Computers & Industrial Engineering*, 2023.

[182] J. Zhou, T. Wu, and X. Li, "Federated learning for cyber-physical systems: Challenges and opportunities," in *IEEE International Symposium on Security and Privacy for IoT*, 2022, pp. 23–32.

[183] J. Kang, S. Liu, and M. Zhang, "Secure federated learning for critical infrastructure networks," *IEEE Access*, vol. 11, pp. 789–805, 2023.

[184] X. Yao, C. Huang, and L. Sun, "Two-stream federated learning: Reduce the communication costs," in *2018 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2018, pp. 1–4.

[185] X. Yao, T. Huang, C. Wu, R. Zhang, and L. Sun, "Towards faster and better federated learning: A feature fusion approach," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 175–179.

[186] L. Rieger, R. M. T. Høegh, and L. K. Hansen, "Client adaptation improves federated learning with simulated non-iid clients," *arXiv preprint arXiv:2007.04806*, 2020.

[187] Y. Yeganeh, A. Farshad, N. Navab, and S. Albarqouni, "Inverse distance aggregation for federated learning with non-iid data," in *MICCAI Workshop on Domain Adaptation and Representation Transfer*. Springer, 2020, pp. 150–159.

[188] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical programming*, vol. 140, no. 1, pp. 125–161, 2013.

[189] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

[190] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[191] R. Ward, X. Wu, and L. Bottou, "Adagrad stepsizes: Sharp convergence over nonconvex landscapes," *Journal of Machine Learning Research*, vol. 21, no. 219, pp. 1–30, 2020.

[192] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[193] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[194] Y. Zhang, L. Huang, and X. Li, "Secure federated learning for hierarchical industrial iot," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, 2021, pp. 156–164.

[195] Microsoft Learn, "What is azure logic apps?" Jan 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/logic-apps/logic-apps-overview