Implementing MLOps on Edge-Cloud Systems: A New Paradigm for Training at the Edge

by

Ridham Dave

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Owing to the rise in data from the Internet of Things (IoT) devices and the increasing demand for intelligent decision-making on the network's edge, there has been a significant surge in interest in the intersection of edge computing, cloud computing and artificial intelligence (AI). Various sectors are adopting such an integrated approach because of the low-latency operating capability due to edge computing, intelligent decision-making due to AI and scalable computing in the cloud. Due to low-latency requirements, in case of performance degradation of the AI application, it is crucial to rapidly adapt and update the edge environment independently while maintaining state synchronization with the cloud.

Owing to the prerequisite for rapid adaptability, a necessity for personalized Machine Learning (ML) training on the edge becomes evident. Furthermore, the universal ML model training is typically conducted in the cloud, leveraging its higher computing resources and abundant data in the central storage. In such a hybrid environment with multiple model sources, it is essential to maintain consistency and a synchronized state of the system. Conventional Machine Learning Operations, also known as MLOps, manage the efficient deployment and monitoring of machine learning models in a single-tier environment.

The challenge of performing MLOps in an edge-cloud environment grows with the number of IoT devices, edge servers and machine learning models. Thus, streamlining the machine learning process, including model training, deployment, and performance monitoring, requires a scalable and robust hybrid approach. To solve the challenge of performing multi-tiered MLOps in a hybrid ecosystem, we propose a novel MLOps architecture to orchestrate the edge-cloud model training and synchronization.

This thesis assesses the proposed architecture using quality attributes, including maintainability, reliability, scalability, functional adaptability and robustness. Furthermore, the thesis tests the proposed architecture in a practical case study experiment, including multiple IoT devices, edge servers and centralized cloud infrastructure. This thesis presents an innovative solution for maintaining ML-enabled edge-cloud systems.

Acknowledgements

I would like to thank my supervisor and advisor, Professor Sebastian Fischmeister, for his continuous support. I would also like to thank my mom and dad for their love and encouragement during these challenging times. Thank you to my friends from Canada and India for believing in me. I would also like to thank everyone from the ESG lab for their continuous motivation.

Dedication

I dedicate this thesis to my family and friends, who have supported me during this journey.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AE Auto Encoders 4

AI Artificial Intelligence 1–3, 6

AIoT Artificial Intelligence of Things 10

API Application Programmable Interface 1

ARIMA Autoregressive Integrated Moving Average 65, 66, 71

ASIC Application-Specific Integrated Circuit 8

AutoML Automated Machine learning 6

AWS Amazon Web Services 4, 5

CCA Cloud Connection Agent 24

CCR Central Container Registry 23, 27, 31, 32

CETS Central Experimental Tracking Server 27, 31

CMLM Central ML Module 25, 27, 30, 31, 33, 45, 51–53, 60

CMR Central Model Registry 27, 30, 31, 34

COE Central Orchestration Engine 24, 25, 30, 44, 52

CSV Comma Separated Values 19

CV Computer Vision 7

DAM Data Acquisition Module 19, 29, 30, 45, 50, 59

DBaas Database as a Service 5

DCU Data Collection Unit 17

DDoS Distributed Denial of Service 7

DevOps Development Operations 8

DIMACE Distributed Machine Learning for Atmospheric Conditions Evaluation in Cloud and Edge 55, 56, 61, 65–71, 73

DTU Data Transmission Unit 17, 18, 44, 47, 48

EC2 Elastic Compute Cloud 4

ECR Edge Container Registry 23, 32

EETS Edge Experiment Tracking Server 22, 32, 34

EMLPM Edge ML Prediction Module 21, 22, 32, 33, 44, 47, 50, 59

EMLTM Edge ML Training Module 20, 22, 31, 32, 45, 51, 53, 59

EMR Edge Model Registry 21, 31, 32, 34, 35

EOE Edge Orchestration Engine 22, 31, 34, 44, 49, 50, 59

ETSM Experiment Tracking Synchronization Module 23, 24, 59

FaaS Function as a Service 5

FPGA Field Programmable Gate Array 50

FR Functional Requirement 12

GPU Graphics Processing Unit 8, 50

IaaS Infrastructure as a Service 5

ICC IoT Connection Client 18, 19, 29, 43, 48, 49

IoT Internet of Things 6, 7, 11–13, 15, 17–19, 22, 26, 29–31, 39–51, 53, 56–62, 64, 68–73

IP Internet Protocol 18

JSON JavaScript Object Notation 19, 57

KDE Kernel Density Estimation 64

MAE Mean Absolute Error 66

MEC Mobile Edge Computing 7

ML Machine Learning 3, 4, 6–10, 12–15, 18–35, 38–42, 44–46, 48–53, 55–61, 64–68, 70, 72

MLOps Machine Learning Operations 2, 8–12, 23, 26, 39–41, 56, 61, 66, 72

MRSM Model Registry Synchronization Module 23, 24, 34, 59

MS Monitoring Server 26, 30, 51

MSE Mean Squared Error 65

NFR Non-Functional Requirement 12, 14, 70

NLP Natural Language Processing 1

PaaS Platform as a Service 5

PCA Principal Component Analysis 4

PSS Persistent Storage Solution 19, 32, 49

SAAM Software Architecture Analysis Method 36–39, 43, 45, 46, 51, 53–55, 71–73

SaaS Software as a Service 5

TM Telemetry Module 23, 31, 49, 59

TPU Tensor Processing Unit 50

UAV Unmanned Aerial Vehicle 7

UDAM Universal Data Acquisition Module 25, 29, 60

Chapter 1

Introduction

Artificial Intelligence (AI) has emerged as a groundbreaking innovation, redefining the boundaries of what humans can achieve. AI has demonstrated its potential in academia, business, art, healthcare and many other sectors [1, 2, 3, 4]. Natural Language Processing (NLP), powered by AI, enables intuitive human-machine interaction, efficient machine translation, and accurate information extraction, among other applications in our digitized world [5, 6]. Recent developments in generative AI models have led to a new form of art and creativity where AI-generated content is indistinguishable from human-generated artifacts [7].

As a cornerstone technology, cloud computing has played a crucial role in the evolution of AI with its on-demand availability of computing infrastructure [8]. This has enabled organizations to process vast amounts of data to build and deploy complex AI models [9]. One significant aspect to consider is that offering AI as a service through Application Programmable Interface (API) has lowered the technical barrier to integrating AI into a solution [10]. Overall, it is evident that cloud computing emerged as a critical catalyst for the emerging developments in AI.

Alongside cloud computing, edge computing has been making strides in recent years by bringing computation closer to the location of data generation [11, 12]. With more internet-enabled devices, edge computing enables real-time decision-making and enhanced operational efficiency. In the context of autonomous vehicles, instead of relaying information to a remote data center, edge computing allows data processing on the vehicle(edge) itself, enabling faster response times [13]. Due to its distributed and independent nature, edge computing saves bandwidth for data transmission and can operate during periods of inconsistent connectivity to the cloud. This parallel rise of edge and cloud technologies

has driven the growth of distributed data processing solutions that cater to a wide range of applications [14].

The latest trends in application architectures showcase an emerging usage of a multi-tier approach, which integrates both cloud and edge computing along with AI capabilities [15]. Such a hybrid approach combines the higher computation capacity and the scalability of the cloud with the localized data processing of edge computing, capitalizing on both technologies' unique benefits. A typical example of such an application could be wearable devices, which can process data regarding vital signs in real-time at the edge and notify the user of any immediate health emergencies. Concurrently, these devices can transmit the data to the cloud for learning long-term patterns to detect potential health issues using AI [16, 17, 18]. By integrating cloud, edge and AI, novel possibilities of intelligent, efficient and real-time systems can be unlocked.

Due to the use of such multi-tier architectures, the need for robust methodologies to manage the life cycle of AI models has increased. Implementing Machine Learning Operations (MLOps) is a general approach in single-tier architectures to ensure efficient deployment, monitoring and updating of the AI models [19, 20, 21]. However, the challenge is to transfer the same concepts to a complex environment, such as a hybrid system with edge and cloud components. Thus, this thesis aims to provide a robust and scalable architecture enabling engineers to apply MLOps concepts in a multi-tier environment.

This thesis offers the following contributions:

- A generalized architecture to deploy, maintain and update machine learning models in a hybrid cloud-edge environment (c.f. Chapter 3).
- A multi-tier system design focusing on automated model management, dynamic functional adaptability, privacy measures, traceability and explainability (c.f. Chapter 3).
- Analysis and evaluation of the proposed architecture's capabilities and limitations (c.f. Chapter 4).
- Experiments and use case to evaluate the quantitative and qualitative metrics regarding the performance of the approach (c.f. Chapter 5).

Chapter 2

Background Information

Understanding the terminology and background information is essential to understand an intelligent multi-tier edge-cloud architecture. This section offers necessary contextual information regarding the scope of the thesis.

2.1 Artificial Intelligence and Machine Learning

AI is a branch of computer science dedicated to studying the augmentation and enhancement of intelligent and intellectual processes using machines, such as reasoning and making predictions![22]. Recently, AI has been engineered to accomplish tasks that require cognitive abilities, such as pattern recognition, speech understanding, autonomous driving and industry automation [23, 24, 25, 26]. Primarily, AI encompasses systems capable of interacting with their environment and making autonomous decisions based on predefined goals. This is often achieved by processing and analyzing large amounts of data to recognize trends and correlations and thus make informed forecasts.

A critical component of artificial intelligence is Machine Learning (ML), a subset that enables these applications to learn and adapt from past knowledge using the data collected from the environment. Machine Learning (ML) involves using mathematical and statistical algorithms to extract insights from data and make predictions or decisions based on the patterns determined by the algorithms [27]. ML possesses extensive applicability across various domains, with its methodologies spanning a spectrum from supervised learning to unsupervised learning techniques [28].

In supervised learning, a labelled dataset is used to train a model(a mathematical function) that serves as a mapping function between the input and the output. Once this function is calculated, the model can generate predictions on new and unseen data. Examples of supervised learning applications include weather forecasting [29], traffic-lane prediction [30], image classification [31], video segmentation [32] and stock prediction [33]. The commonly used techniques in supervised machine learning include linear regression, logistic regression, decision trees, random forests, k-nearest neighbours, AdaBoost and neural networks [34].

On the other end, unsupervised learning works on datasets without historical labels. However, instead of predicting outcomes, it focuses on finding hidden structures and patterns in the unlabeled data. Furthermore, this methodology provides a novel direction for discovering patterns during the exploratory data analysis stage [35]. There are two types of unsupervised learning techniques: dimensionality reduction and clustering. Dimensionality reduction uses algorithms like Principal Component Analysis (PCA) [36, 37] and Auto Encoders (AE) [38, 39] to obtain a subset of principal variables, which can help to visualize the data in lower dimensions and remove noise and randomness from the dataset. In addition, clustering algorithms like k-means [40, 41] and hierarchical clustering [42, 43] are used to divide a given dataset into clusters such that the data instances in the same cluster follow a distinguishable pattern and have similar features. Examples include clustering websites based on raw text and grouping customers based on purchase history.

Between these two extremes lies a vast range of ML techniques: self-supervised [44], semi-supervised [45], and reinforcement learning [46]. These techniques borrow concepts from supervised learning, unsupervised learning and the notion of reward function for cumulative reward.

2.2 Cloud Computing

Cloud computing has emerged as a mainstream technology in the last decade, primarily influenced by the competition between the tech giants like Amazon and Rackspace [47]. Amazon entered the cloud computing space with its offering of Elastic Compute Cloud (EC2) with its launch of Amazon Web Services (AWS), which offered scalable computing solutions in the cloud. This paradigm shift provided businesses access to flexible and scalable resources, which were initially only affordable to large corporations. Thus enabling more accessible web computing infrastructure access for developers and smaller businesses without investing in expensive hardware or a data center.

Economically, cloud computing provides significant cost-effective alternatives to businesses, which boosts competition in the industry and promotes entrepreneurial initiatives in a large variety of sectors. Furthermore, it is essential to note that cloud computing is reshaping the business models for startups, which can, in turn, focus on core technologies and competencies [48]. Current cloud platforms such as AWS, Google Cloud, Microsoft Azure and IBM provide a pay-as-you-go model, which allows companies to only pay for the resources they consume [49, 50]. Thus eliminating the upfront capital expenditure and reducing the total cost of ownership.

The offerings by cloud providers can be broadly categorized into the following services:

- Infrastructure as a Service (IaaS): IaaS facilitates raw computing resources like compute machines, storage solutions and network infrastructure, where the highest level of management control and flexibility is provided to the user. Prominent examples include Amazon EC2 and Google Compute Engine [51].

- Platform as a Service (PaaS): PaaS refers to a cloud computing model that offers developers a framework they can build upon by abstracting infrastructure management [52]. AWS Elastic Beanstalk and Google App Engine are examples of PaaS services.

- Software as a Service (SaaS): SaaS refers to a cloud-native platform where a third-party company hosts applications for customers over the internet on a pay-as-you-go revenue model. This eliminates the customer's need to manage or maintain the software or hardware [53]. An example of this would be Salesforce and Google Workspace.

- Function as a Service (FaaS): FaaS enables the companies to run and manage applications and their corresponding functionalities without maintaining the underlying infrastructure [54]. The cloud provider manages the necessary computing resources, where the pricing is based on consumption, allowing the user to focus on the system's functionality. Some popular FaaS services include AWS Lambda functions and Microsoft Azure Functions.

- Database as a Service (DBaas): DBaas refers to a fully managed cloud computing service that provides users access to a database without managing any underlying infrastructure. It enables users more accessibility to set up policies for backup, scaling and replication [55]. AWS provides multiple solutions for DBaaS, including Amazon RDS and Aurora Serverless.

Machine Learning in the Cloud

The emergence of Machine Learning (ML) services in the cloud has sparked a significant rise in AI usage. Various ML-focused tools on the cloud have enabled individuals to integrate advanced AI models into their architectures by abstracting the complexities of infrastructure management and scaling. Some popular cloud services that provide fully managed service for developers and data scientists include Amazon Sagemaker, Google Cloud AutoML, Microsoft Azure Machine Learning and IBM Watson Studio [56].

Furthermore, modern cloud platforms provide pre-built and ready-to-use ML models, lowering the experience needed to develop intelligent and advanced systems. Microsoft Azure provides access to the state-of-the-art OpenAI's GPT models through its Azure OpenAI Service [57]. In addition, the latest advancements in Automated Machine learning (AutoML) in the cloud suggest making complex ML models accessible to a broader audience. AutoML provides the capabilities to automate the ML process using user-friendly interfaces and highly scalable workflows [58].

2.3 Edge Computing

Edge computing is often described as a federated computing model that allows computation at the network's edge. It involves processing sensor data closer to the Internet of Things (IoT) devices instead of transmitting it to the cloud for processing. This enables applications with high-speed, low-latency requirements to be deployed near the data sources.

In the past few years, an unprecedented upsurge in edge devices has been observed in fields like transportation, retail, defence, home automation, and healthcare [59, 60]. Forecasts predict that over 75 billion IoT and edge devices will exist across multiple sectors by 2025 [61]. In addition to the increase in the number of edge devices, the computational capabilities of individual devices are also on an upward trajectory. Modern edge devices boast more computational power, storage capacities and AI capabilities than their earlier versions.

Edge computing significantly reduces the latency for real-time applications compared to cloud computing using localized data processing. Bringing data processing closer to the edge device eliminates the need to transfer data over extended distances to a remote cloud server, thereby minimizing the noticeable delay. In autonomous vehicles and healthcare, this ensures safe and effective operation through real-time data processing and faster decision-making [13].

Another fundamental benefit of edge computing is its ability to reduce bandwidth requirements. In the traditional cloud paradigm, all data from the IoT sensors are transferred to the centralized cloud server for processing, which can strain the network resources. Edge computing's efficient approach mitigates these bandwidth issues by local processing of data, which can include aggregation and filtering of data on the edge device. This approach offers cost benefits where bandwidth usage incurs significant costs [62].

Edge computing offers several advantages over conventional cloud models regarding privacy and security. The edge computing model decentralizes data storage and processing on edge devices, allowing personal and sensitive data to persist on the edge itself, thereby reducing data transfer to the cloud [63]. Furthermore, as data processing happens on the edge device, the system's vulnerability against cyber-attacks during data transmission opportunities is reduced [64].

Machine Learning on the Edge

With the immense quantity of data produced by IoT devices, extracting information from raw data and making intelligent decisions using the limited computing capacity on the network edge is essential. One promising approach to address this issue involves using TinyML [12], which refers to applying ML on edge. This includes inference and often training the ML model on edge in a resource-constrained environment.

Machine Learning (ML) on edge has several benefits ranging from faster response times, increased privacy, hyper-personalization, reliability and reduced bandwidth usage. In the intelligent agriculture domain, researchers leveraged edge computing to study the learning factors influencing plant growth patterns [65]. Recent research conducted by [66] compares the performance of ML algorithms in Mobile Edge Computing (MEC) to detect Distributed Denial of Service (DDoS) attacks.

A study by Cristian Toma et al. [67] presents the challenges in implementing ML solutions in an edge environment, ranging from applications in Unmanned Aerial Vehicle (UAV) Drones, Robotics and IoT devices. Madhavi et al. deployed a machine learning-based granite classifier on an ESP8266 board, demonstrating the capability of a microcontroller to run ML models to detect different types of granite [68].

In Computer Vision (CV), ML on edge can enable the analysis and interpretation of visual signals directly near the digital sensor in real-time. Ananthanarayanan et al.'s video analytics system [69], Rocket, efficiently processes video data from various cameras and produces highly accurate outputs in a resource-constrained environment. Wang et al.

devised a video analytics framework leveraging the edge computing model that allows an efficient deployment of ML applications on autonomous drones [70].

Machine Learning (ML), coupled with edge computing, is playing a pivotal role in autonomous vehicles. Hochstetler et al. perform real-time object detection using a popular deep-learning development kit, Intel Neural Compute Stick, and a Raspberry Pi [71]. STTR [72], an intelligent vehicle tracking system, can handle multiple cameras feeds in real-time to calculate the vehicle's trajectories, thereby minimizing the data stored in the edge device.

Significant strides have been made toward accelerating edge machine learning in recent times due to the development of specialized hardware and algorithmic optimizations. Regarding hardware, the rise of specialized devices like Graphics Processing Units (GPUs) and Application-Specific Integrated Circuits (ASICs) has accelerated parallel edge computations. Certain variations of ASICs and GPUs have been optimized for edge environments to deliver high computational power while maintaining low power consumption, enabling the adoption of complex applications like video segmentation and image super-resolution on the edge.

On the algorithmic front, multiple studies have demonstrated significant improvement in the operational efficiency of ML models on the edge. SquuezeNet [73], a deep learning network architecture, performed equally well as AlexNet [74] on image classification tasks, even with 50 times lower parameters to learn. It achieved such performance by aggregating the concepts of late down-sampling in the network, using a higher number of 1x1 filters and updating convolutional layers with fewer output channels.

2.4 Machine Learning Operations

Machine Learning Operations (MLOps) leverage Development Operations (DevOps) principles to integrate ML models into a software system efficiently. MLOps aims to streamline multiple ML steps ranging from model development and quality assurance to model delivery and monitoring into a single process [75]. It aims to maintain reliability, robustness and reproducibility in a ML system by standardizing various development and operations practices [76, 20].

MLOps has the potential to bridge the gap between data scientists and IT professionals by integrating Continuous Integration and Continuous Deployment (CI/CD) pipelines in the development process, thereby accelerating ML deployment timelines [77]. MLOps supports the seamless transition of ML models from development to quality assurance

through automated pipelines for data and model validation during the CI stage. In addition to that, CD pipelines enable the reliable and rapid deployment of validated models to the production environment.

A typical MLOps lifecycle can be divided into these fundamental steps:

- Data Ingestion: This step involves aggregating data from multiple sources and transforming them into the formats necessary for ML models.
- Model Development and Training: It refers to the designing ML models by data scientists and ML engineers and the models training using prepared datasets.
- Model Validation: It involves testing and validating the ML model on unseen data, evaluating its performance and estimating its accuracy in the real world.
- 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 change of hyperparameters on the model.
- Model Versioning: This step keeps track of validated models and ensures traceability and accountability in ML workflows.
- Model Deployment: Once the model is approved and ready for delivery, this step deploys the model to the production environment using continuous delivery pipelines.
- 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.
- 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.

Multiple open-source and commercial MLOps tools provide capabilities to streamline the ML life cycle. MLFlow, an open-source platform [78], provides experiment tracking, model registry and model serving capabilities [79]. Similarly, Kubeflow [80] is a scalable way to deploy ML workflows in a Kubernetes environment, where each workflow step is wrapped inside a container. Another popular MLOps tool is TensorFlow Extended (TFX) by Google [81], which provides a framework for data validation, model training and model serving. Amazon Sagemaker, Azure ML and GCP Vertex AI are examples of commercial cloud ML services which provide a range of services for the ML lifecycle [82].

MLops in an Edge Environment

Edge MLOps is an emerging field in software engineering that lies at the intersection of ML, DevOps and Edge Computing. MLOps principles help define a repeatable, reliable and maintainable approach for managing a scalable edge ML environment. It provides a framework to address various challenges, such as device resource constraints, hardware heterogeneity, dynamic adaptability across various environments and model monitoring in a distributed setting [83].

Many edge applications are deployed on embedded devices with limited computing and storage capacity across various domains. To deploy these ML models in a resource-constrained environment, the ML models need to be optimized for complexity and resource usage [84]. Multiple studies have focused on improving deep learning architectures for edge devices, thereby allowing complex models to run on microprocessors and microcontrollers [85, 86, 87]. Building upon prior research, Tao and Li proposed a new optimization method to reduce the gradient size up to 90% during model training in a convolution neural network architecture without reducing the convergence rate [88].

Multiple MLOps frameworks have been designed to streamline the ML lifecycle at the edge. Raj et al. proposed a novel edge MLOps framework for Artificial Intelligence of Things (AIoT) to automate continuous model training, deployment, delivery, and monitoring for ML models while utilizing the Microsoft Azure platform for the management of IoT devices [89, 90]. SensiX++ provides a dynamic runtime for ML model inference on various edge devices with an integrated MLOps module [91]. It allows serving multiple models on edge devices with granular control while minimizing repeating data operations, thus automating MLOps steps in an edge environment.

The Tiny-MLOps framework aims to bring ML capabilities to low-energy 32-bit microcontrollers with stringent resource constraints [83]. The experiments in the study, using data collected from various sensors monitoring an industrial rotary machine, provide evidence of the feasibility of the edge MLOps framework. Edge Impulse [92] is another framework that can support optimizing ML models for various embedded edge systems. Furthermore, Pangea is an MLOps tool that creates a robust runtime environment for edge devices to run custom analytical pipelines [93].

Chapter 3

Architecture

In computer engineering, system architecture refers to a system's high-level structure and design. It may include multiple hardware devices, software modules and their connections. Architecture provides a high-level understanding of the underlying hybrid system [94]. Apart from the components, a system architecture provides a blueprint of the system's behaviour under different scenarios.

System architecture serves as a blueprint for a digital system, where the components include all hardware and software elements [95]. For example, an IoT device in an edge environment and a web application hosted in a cloud environment can be considered components of a hybrid system. Apart from listing these components, inter-component and intra-component relationships are crucial to any architecture. These relationships include the communication and transactions between multiple system components. An architecture further characterizes the system's behaviour, describing how various modules deal with external inputs like user click and webhook trigger. Lastly, it encapsulates essential system properties, referred to as quality attributes, including scalability, security, maintainability, reliability and performance [96, 97].

Designing a system architecture requires formulating a set of functional and non-functional architectural requirements [98]. Defining a system's precise requirements is essential in the system development life cycle. This chapter formalizes the requirements of an edge-cloud MLOps system and provides a functional block diagram of the underlying architecture. Lastly, a behavioural view of the architecture is described using the primary use-case scenarios of the system.

3.1 Requirements

Requirements help estimate the project's scope while also providing a clear definition of the system's expectations. To guide the system's design, requirements are classified into two main categories: functional and non-functional [99]. Functional Requirement (FR) are the fundamental requirements for the system's behaviour, operations and activities. For example, data collection and transmission are functional requirements for an edge IoT system. On the other hand, Non-Functional Requirement (NFR) refer to the quality of the function execution rather than the function itself. In other words, non-functional requirements define how a system performs its functional operations. Some of the non-functional requirements of an IoT system include scalability, reliability, security and energy efficiency.

Designing an optimal system requires thoroughly understanding the requirements and finding potential tradeoffs. Tradeoffs are decisions where improvement in one performance metric can degrade the performance of another performance measure [100]. The primary focus of any system design study is to find the perfect balance between these compromises, achieving optimal performance and fulfilling the required functional and non-functional requirements.

The motivation of this study is to understand the conflicting requirements of the system, optimize the management of the underlying tradeoffs and build a scalable edge-cloud MLOps architecture for a hybrid environment.

3.1.1 Functional Requirements

Functional Requirement (FR) specify the system's functionality, including its operations, behaviour and properties [101]. In addition, FR plays a pivotal role in defining the use cases of a system. Such use cases help to describe the system's behaviours to achieve specific goals, determine the primary components of a system and describe how different components interact with each other. Lastly, use cases assist in identifying integration points of system components with external systems.

The core functionality of the proposed edge-cloud MLOps architecture is to seamlessly train, deploy, monitor and update ML applications in an edge environment, leveraging cloud infrastructure.

Each functional requirement is assigned with an abbreviation FR, appended with a number for reference, and then a short description detailing the requirement. Furthermore,

each instance of the functional requirements is derived from the corresponding cited study to which it is linked via citation. The following outlines the requirements:

FR-1 *The system shall facilitate data collection from IoT devices.* [102]

FR-2 *The system shall support data storage on the edge device.* [103]

FR-3 *The system shall support data collection from public data sources.* [104]

FR-4 *The system shall support data storage on the cloud infrastructure.* [105]

FR-5 *The system shall support communication between IoT devices and edge devices.* [106]

FR-6 *The system shall support synchronization between edge devices and cloud infrastructure.* [107]

FR-7 *The system shall allow seamless access to data on the edge device.* [108]

FR-8 *The system shall facilitate centralized access to data on the cloud infrastructure.* [108]

FR-9 *The system shall support the universal training of the ML model in the cloud environment.* [109]

FR-10 *The system shall support the personalized training of the ML model on the edge device.* [110]

FR-11 *The system shall support model inference on the edge device.* [111]

FR-12 *The system shall log and monitor the performance of ML models on the edge device.* [112]

FR-13 *The system shall support the synchronization of ML training experiments and models from the edge to the cloud.* [107, 113]

FR-14 *The system shall support the storage of the ML model and experiments in the edge and the cloud.* [114]

FR-15 *The system shall facilitate centralized access to the global and individual performance of the ML models.* [115]

FR-16 *The system shall support over-the-air updates of components.* [113]

FR-17 *The system shall support detecting the degradation of performance of an ML model.* [116]

FR-18 *The system shall support universal and personalized retraining of a model.* [117]

FR-19 *The system shall support updating the ML model in the edge and cloud environment.* [87]

3.1.2 Non-functional Requirements

While functional requirements provide a detailed description of the behaviour and operation of a system, NFRs dictate the overall characteristics of the behaviours and operations [118]. They provide a metric to determine the system's performance and ensure its quality, scalability, reliability, availability, and maintainability are adequately defined [119].

Similar to the format defined in Section 3.1.1, non-functional requirements are denoted with an NFR abbreviation, followed by a number. Again, the non-functional requirements are derived from the studies cited along with each requirement instance.

NFR-1 *The system shall support simultaneous training on three ML models on the cloud and one ML model on the edge.* [120]

NFR-2 *The system shall maintain hot storage for model training data for 24 hours on edge.* [17]

NFR-3 *The system shall facilitate a data archiving policy for universal data on the cloud for a minimum of 30 days.* [121]

NFR-4 *The system shall support inference on the edge device with maximum 500-millisecond latency.* [111]

NFR-5 *The system shall support synchronization of the ML model and experiment from the cloud to the edge in under 1 minute.* [107]

NFR-6 *The system shall support monitoring metadata synchronization with a maximum latency of 10 seconds.* [107]

NFR-7 *The system shall support over-the-air updates of the ML model within 2 minutes from the cloud to the edge.* [87]

NFR-8 *The system shall support over-the-air update of edge components within 3 minutes.* [87]

3.2 Functional View

In a system design, a functional view of a system provides details about the structure of an architecture. The functional view is a system perspective that focuses on the functionalities of individual components [122]. In addition, a functional view formalizes the architecture using a functional block diagram that provides the stakeholders with a clear and understandable way to describe the working of a system. Lastly, a functional view can provide a clear set of criteria to conduct system testing to evaluate the system's performance.

Figure 3.1 represents a functional block diagram of the proposed ML architecture in a hybrid environment. This diagram enables the removal of abstractions and provides proper guidelines for components and their necessary connections. Each box in the architecture refers to a component or a sub-component in the system, while each line connecting individual boxes represents the interaction between those components. These interactions include data transmission, log synchronization and command routing.

The architecture consists of two different environments, edge and cloud environments, with proper separation provided by the dashed line. The edge infrastructure comprises the edge server and the IoT devices, while the cloud infrastructure consists of various components to orchestrate and manage the system centrally [123].

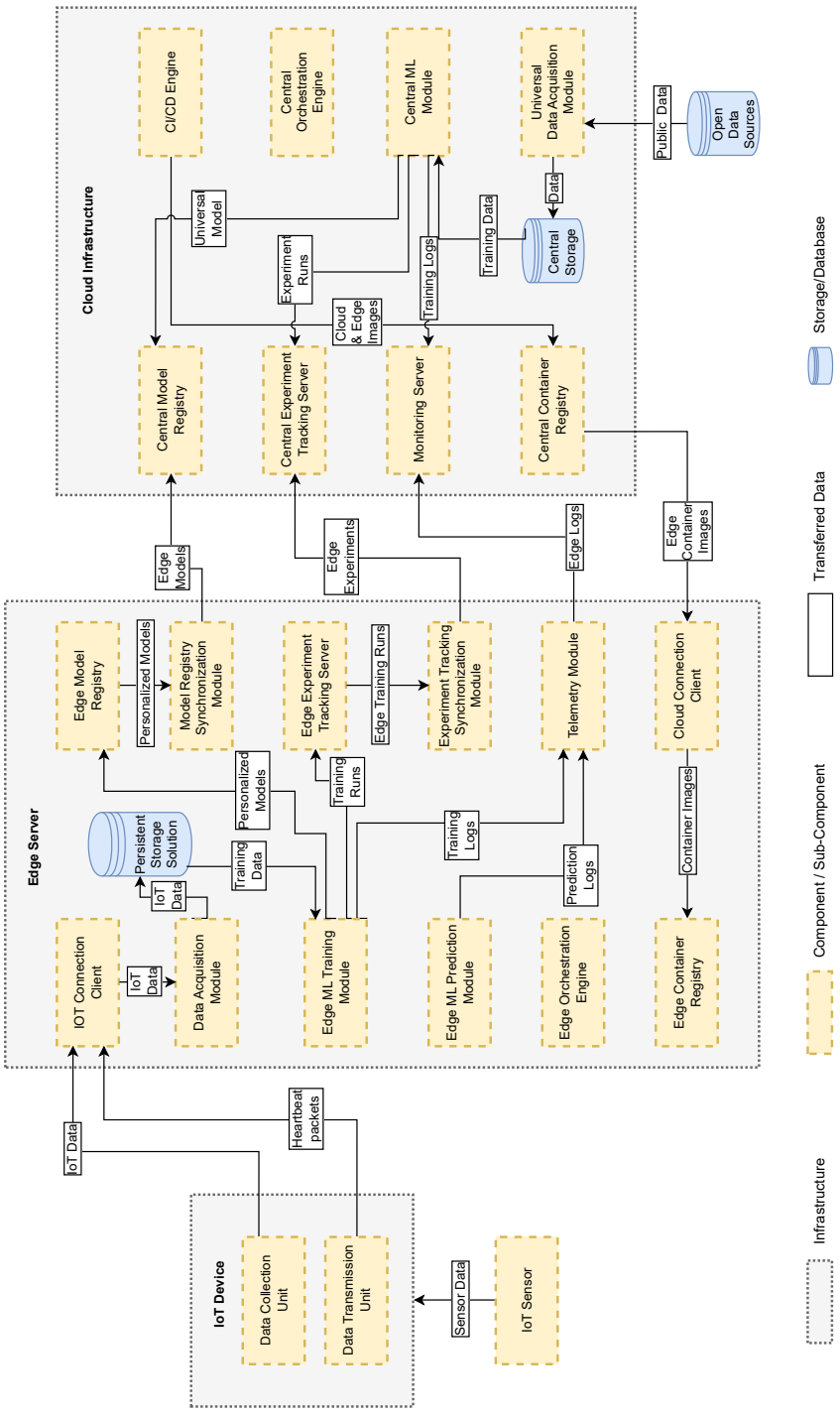


Figure 3.1: Functional Block Diagram

Subsequently, a breakdown of each component, along with its sub-components and functionalities, is provided below.

3.2.1 IoT Device

An Internet of Things (IoT) device plays an integral role in the data acquisition pipeline of any edge system [124]. It is responsible for interfacing with the physical world and is typically embedded with multiple sensors. In the overall architecture, these devices act as data sources and the first level of data processors. Furthermore, these devices support real-time data collection using power-efficient embedded architecture. Lastly, these devices also have communication hardware to transmit data to the upstream infrastructure.

Data Collection Unit (DCU)

The primary function of a Data Collection Unit (DCU) is to capture data from the device's surrounding environment continuously, as required by FR-1. Such data collection often uses embedded sensors that measure environmental attributes like temperature, light intensity, pressure and humidity. Apart from collecting raw data, the module is also responsible for processing the initial data, including calibration based on equipment and measurement unit transformations. Multiple studies suggest the use of a dedicated component to collect data, thus enhancing modularity and overall system reliability and security [125, 126, 127, 128, 129].

As IoT devices are capable of generating massive volumes of data, the DCU has the ability to control the resolution and frequency of data capture, thus, controlling the overall bandwidth requirement of the system. Hence, the DCU must be designed to handle the change in data capture frequency.

Data Transmission Unit (DTU)

Another crucial component of an IoT device is the Data Transmission Unit (DTU) as demonstrated in various studies [130, 131, 132, 133, 134]. This module is responsible for bi-directional data transfer between the edge server and the IoT device. Also referred to as the communication module, this module handles the transmission of messages using various communication protocols, as required by FR-5.

The DTU packages the collected data into necessary transmission formats and includes data serialization, data integrity parity bits, and encryption in the message payload. For

example, DTU can generate an Internet Protocol (IP) packet for communication over the internet and can also support data transfer using Bluetooth for shorter-range applications. Furthermore, the DTU often incorporate security measures, including encrypting the data based on the use case. In addition, the DTU is capable of transmitting a status packet to the edge server containing a heartbeat message and device metadata [130, 135].

Lastly, the DTU also supports receiving messages from the edge server, including control messages to manage the DTU [134]. These control messages include triggers to start/stop data collection, start/stop data transmission and update the data capture frequency.

3.2.2 Edge Server

In an edge machine learning system, the edge server, also known as the edge node or edge gateway, handles critical tasks ranging from local processing and storage of sensor data to ML training and inference workflows [136, 137]. As the data from the IoT infrastructure are sensitive and personal, the edge server serves as an intermediate node that stores data locally and preserves the privacy of the underlying data [138, 139].

The edge server should have sufficient computing power to handle data processing, ML workloads and network functions [140, 141]. In addition to that, the edge server requires adequate storage to store persistent and temporary data for ML model training. Finally, the edge server functions as a bridge between IoT devices and the cloud; hence it should have robust network capabilities to interact with multiple network nodes simultaneously [142, 143].

Figure 3.1 presents the structure of the edge server within a distributed network. The subsequent sections detail each sub-component within the edge server, along with comprehensive use cases.

IoT Connection Client (ICC)

The IoT Connection Client (ICC) manages connections with multiple IoT devices in a distributed edge environment [144, 143, 142]. Furthermore, the ICC continuously monitors the status of the connected devices, along with tracking the operational parameters of the IoT device and its connection, as required by FR-5 [145].

The ICC handles the connections between the IoT device and the edge server by establishing, maintaining and managing connections depending on the connection methodology and network topology. The ICC supports multiple communication protocols and deals with

connection interruptions and failures. Furthermore, the ICC monitors crucial operational parameters, including data regarding computing loads, transmission rates and battery level of the IoT device. Lastly, ICC provides necessary meta-data about the connection status of an IoT device, enabling robust error detection and handling [138].

This module is critical in scaling the IoT infrastructure as it handles simultaneous connections with multiple IoT clients, allowing infrastructure growth. The ICC is designed to handle the increasing number of IoT devices without substantially affecting the overall system's performance as demonstrated my multiple research works [145].

Data Acquisition Module (DAM)

The sub-component Data Acquisition Module (DAM) handles the streaming data from the IoT devices that are connected to the edge server[141, 146, 147]. The module is tasked with decoding and unpacking data packets received from the IoT device and validating the data using integrity information present in the message payload. The DAM processes the input data, which involves cleaning the data, transforming the data and converting the units of measurement if required.

During this process, the DAM stores the incoming data in the onboard persistent storage for machine learning workflows, as required by FR-2. Such local storage capability eliminates the need for transferring data to the cloud, thereby reducing latency and bandwidth usage and enhancing privacy and security. The DAM supports multiple data storage adapters, allowing a seamless change in the database type according to the application's needs[148, 139]. For example, the DAM can store incoming data in SQL storage by supporting various connectors for different database solutions. Lastly, the DAM can interface with object storage solutions to save the raw data in binary, JavaScript Object Notation (JSON) or Comma Separated Values (CSV) format.

Persistent Storage Solution (PSS)

Persistent Storage Solution (PSS) refers to the data retention component on an edge server to store IoT data for future ML workflows[139, 129]. Unlike temporary storage, PSS is independent of individual processes and survives system reboots, as required by FR-2 and FR-7. Furthermore, it also supports the implementation of life-cycle rules for the data due to its limited storage capacity.

In the case of a SQL solution, custom data management events can be configured that automatically delete the data after a certain period. On the contrary, standard object

storage solutions usually provide built-in capabilities for life cycle policies that enable the deletion of objects after a specific interval.

Edge ML Training Module (EMLTM)

The edge ML training process refers to the workflow of leveraging the data stored on the persistent data storage on the edge server to train an updated machine learning model [149, 141, 144]. It leverages the computing resources of the edge server to train models closer to the source of data, thereby preserving privacy and enhancing data security, according to FR-10. In the case of model degradation of an already deployed ML model, Edge ML Training Module (EMLTM) can train a new model or refine an existing model to improve prediction performance [150].

The edge training capability allows the edge server to adapt to a decline in performance due to data drifts, allowing the new and updated model to replace the older model with reduced latency [141, 151]. Thus, EMLTM allows the edge system to operate in scenarios where the network connectivity with the cloud infrastructure is unreliable.

The significant functionalities of the module include logging and tracking each training run and the trained model for reproducibility, traceability and benchmarking [152, 153, 154]. On receiving a trigger from the upstream orchestration, the EMLTM initiates the data fetching stage, which queries data from the persistent storage. Next, the module processes the input data according to the model's needs and initiates the training process. Finally, the trained model is benchmarked and validated against past data to capture its training and testing accuracies.

The module plays a pivotal role in tracking the training process, which involves logging the following information:

- Training initialization trigger details
- Data utilized for the training and testing step, along with the input queries
- Hyperparameters used for training the model
- Training and testing accuracy
- Meta-data for the training process, for example, training time, CPU usage
- Training logs

The above-listed information is encapsulated as an experiment and saved in an experiment tracking module present on the edge server. Finally, the trained model is logged in the model registry and registered as ready for deployment on the edge server. The ML model, saved in the registry, is tagged with an experiment id extracted from the tracking server. It is crucial to reference the model and the experiment together to trace the model's origin and thus reproduce the model and verify the results if necessary.

Edge ML Prediction Module (EMLPM)

The Edge ML Prediction Module (EMLPM) is responsible for deploying and serving the latest ML models on the edge device [155, 111]. This module ingests raw input data, processes it, and uses a trained ML model to make predictions, as per FR-11. In addition, the EMLPM supports the life-cycle management of the model, which allows updating the models when a newer version is available [156, 157].

The prediction module facilitates preprocessing of the raw data, often including feature extraction, normalization and standardization [158]. The module uses the best-performing model in the edge model registry for prediction and, lastly, logs the predicted values to the local storage. These predicted values assist in calculating the performance of a deployed model when compared against labelled data, as required by FR-12.

The crucial functionality of this sub-component is to fetch the appropriate model from the model registry and serve the model for prediction use-case. This functionality includes continuously checking if a newer model is available and thus replacing the served model with it. Finally, EMLPM also supports the capability for batch prediction, which plays a vital role while benchmarking past models using new data.

Edge Model Registry (EMR)

Tracking the trained models and storing the previous models in persistent storage is crucial for any ML system [159, 160]. The model registry is an artifact repository that manages and tracks ML models in any environment. In this edge architecture, the Edge Model Registry (EMR) provides functionality to independently version the models, track necessary metadata, and transition models between different stages, as required by FR-14 [161, 154].

The EMR keeps track of various ML models on the edge storage, providing the ability to compare various models to fetch the best model. Apart from that, rollback, in case of an anomaly in the training process, to a previous model is allowed using this functionality.

Edge Experiment Tracking Server (EETS)

The Edge Experiment Tracking Server (EETS) systematically tracks information about each ML training run, also known as an experiment, to the edge storage. The critical role of this sub-component is to record the detailed account of each step during a training process, which includes input data, training hyper-parameters, model accuracy and logs [162, 163, 164]. Furthermore, the tracking server also supports comparing previous experiments to select the best model training process, enabling guidance while selecting hyper-parameters for future experiments.

Edge Orchestration Engine (EOE)

The Edge Orchestration Engine (EOE) is the central element of the edge architecture that manages the overall execution of other sub-components on the edge server [165, 138]. In an Edge system, the primary responsibilities of an EOE are workflow management and task scheduling. The EOE coordinates the internal processes for IoT device management, life-cycle management of persistent storage, model validation trigger, machine learning retraining trigger and system resource management. Apart from the internal tasks, the engine is also responsible for orchestrating the synchronization of models and experiments to the cloud and updating components in the edge based on triggers from the cloud infrastructure [140, 144].

The EOE controls the sequence of steps for any workflow in the edge environment, including triggering and monitoring the EMLTM and EMLPM. The engine supports multi-step workflows that include decision-making steps and parallel processing [144]. For example, a workflow for model life-cycle management can include data processing and model prediction as the first two steps, and the third and the fourth steps to retrain the model and update the model is only executed based on the output of the second step, the model prediction step, according to FR-17.

Another aspect of the EOE is the scheduler, which is responsible for triggering, monitoring, and updating tasks based on predetermined intervals. For example, steps to delete legacy logs, update life-cycle policies, trigger system clean-ups and synchronize with the cloud are configured as cron jobs in a scheduler [158].

Model Registry Synchronization Module (MRSM)

As discussed earlier, training an ML model on edge has multiple benefits, but it is crucial to synchronize the models with the cloud regularly. The aggregated learning from different

edge devices empowers the training of a better-aggregated model while preserving data privacy in a distributed federated environment. Also, keeping track of various legacy models is essential to any MLOps operation for governance and reproducibility.

The Model Registry Synchronization Module (MRSM) enables the edge to transfer models to the cloud infrastructure for backups. In addition, the MRSM is also responsible for downloading the universal model trained in the cloud to the edge device, resolving the cold-start problem, as required by FR-19. The cold-start problem in an edge ML system refers to the new deployment of an edge device where past data is not available to train personalized models.

Experiment Tracking Synchronization Module (ETSM)

Like the MRSM, the Experiment Tracking Synchronization Module (ETSM) connects to the cloud infrastructure to upload the local training experimental runs. Uploading the experiments to the cloud can help leverage the power of collaborative retraining of the universal model while still harnessing the privacy of edge computing. Furthermore, this component allows experimental data offloading to the cloud, thus allowing the reuse of the storage space on an edge system, as required by FR-13.

Telemetry Module (TM)

The Telemetry Module (TM) collects operational data from various sub-components and forwards them to the cloud infrastructure [166, 158]. Real-time logging and monitoring enable the cloud system to track the edge servers and detect any operational issues or anomalies [167]. In addition, the logging module maintains an audit trail, also referred to as a record of steps or changes in the edge system. This enables understanding past system behaviour and assists in debugging any issues or bugs encountered in the edge environment [147].

Edge Container Registry (ECR)

The Edge Container Registry (ECR) is a repository for container images containing code for various system modules' underlying functionality inside an edge server [168]. A ECR enables a seamless transition of a sub-component from one version of a component to another, allowing faster updates and rollbacks [157, 169, 170]. Furthermore, a container

registry provides storage for container images, enabling reproducibility and auditing capability [171]. Lastly, the ECR synchronizes regularly with the Central Container Registry (CCR), enabling quicker updates.

Cloud Connection Agent (CCA)

The Cloud Connection Agent (CCA) enables two-way connection and synchronization between the cloud and the edge server, as required by FR-6 [14, 172]. It is responsible for ensuring reliable and secure connectivity with the cloud, enabling data and command transfer between edge and cloud infrastructure [173, 174].

The CCA ensures the synchronization facilitated by the MRSM and ETSM is consistent and keeps the overall system in coordination. Furthermore, the CCA manages the relay of commands from the cloud layer to the edge, enabling timely updates and configuration changes. In conclusion, the CCA functions as a bridge between the edge server and the cloud infrastructure.

3.2.3 Cloud Infrastructure

The architecture's cloud infrastructure serves as the hybrid system's central controller, responsible for managing the edge servers and ensuring that all system elements work seamlessly together [175, 14]. The cloud infrastructure can be deployed directly on a virtual machine in the cloud or as a distributed managed service, hosting individual components in separate compute instances [176]. Furthermore, the storage solution, either object storage or relational databases, can be deployed separately using built-in cloud services or open-source applications.

In the cloud environment, various pipelines for deploying the latest code, models and configurations can be triggered to update the edge server [16]. In addition to that, the pipeline coordinates the seamless code integration and code delivery for any new system changes. On the machine learning front, the cloud trains the universal model and keeps track of individual models deployed on each edge device. Each ML workflow on the edge is monitored in the cloud, providing a real-time global system view [158].

Central Orchestration Engine (COE)

The Central Orchestration Engine (COE) acts as a primary hub for coordinating the cloud's various components, which are, in turn, responsible for managing the hybrid edge-cloud

environment [175, 156]. Similar to the edge server, the primary engine in the cloud infrastructure is divided into two sub-components: the workflow manager and the scheduler.

From ML perspective, the COE triggers the initial workflow for universal model creation, which internally invokes data acquisition from the public data sources and then initiates the machine learning training step [15, 177]. Similarly, the COE manages the retraining logic for updating the universal model based on newer data from public sources or aggregating the knowledge from the personalized edge models.

Furthermore, the COE is responsible for scheduling chronological triggers to update the cloud and edge systems, trigger system clean-ups, and update life-cycle rules to move the universal data from hot to cold storage. The COE also supports a configuration interface, which enables the system administrator to update the run-time parameters like cron intervals, system performance thresholds and alert configurations.

Universal Data Acquisition Module (UDAM)

To train a universal machine learning model, also known as the global model, it is crucial to collect data for the initial model training [178, 179]. The key to any accurate global model is to train on a dataset which is as similar as possible to the data that model will encounter in the deployment environment. This is because an ML model generalizes on the training dataset and uses the distilled knowledge to predict unseen data.

To collect such data, the Universal Data Acquisition Module (UDAM) leverages publicly available datasets and processes them depending on the use case, implementing FR-3 [180, 181]. Furthermore, the UDAM saves the scraped data to the central persistent storage for further use by the ML module, as required by FR-4 [182].

To comply with the relevant data protection and privacy laws, care should be taken to collect data without Personally Identifiable Information. Hence, privacy-preserving techniques like anonymization and differential privacy are often applied along with this step [179].

Central ML Module (CMLM)

The Central ML Module (CMLM) efficiently encapsulates the functionality of both the training and the inference modules in the cloud environment [82, 183]. The CMLM is responsible for running the ML workloads on the data present in the persistent storage while interfacing with the central orchestration engine [184].

One such workflow involves training the universal model based on data collected by the UDAM, which internally includes preprocessing the data, training the model, and logging the experiments and the models to the respective tracking servers, as required by FR-9 [175]. Another workflow managed by the CMLM is the flow for retraining the universal model by aggregating the knowledge distilled by the personalized edge models using the concept of federated learning.

Finally, this module also assists in calculating the prediction accuracy of the individual models, the universal and the personalized models, using the unseen test data.

CI/CD Engine

The Continuous Integration and Continuous Deployment (CI/CD) engine is vital in the MLOps process, especially in automating the software and machine learning delivery process [185, 186]. The CI/CD engine manages any modern architecture's integration, testing, delivery, and deployment steps [187].

The pipelines include coordinating with the central ML module and deploying the updated universal model in the edge server for the ML delivery process [188]. In addition to that, the CI/CD engine orchestrates the pipeline of extracting knowledge from the personalized models and utilizing the knowledge to train the new universal model [186].

From a software development perspective, the CI/CD engine coordinates the integration and testing of new code changes, enabling higher code quality [157]. Once the integration tests succeed, the continuous delivery builds the necessary artifacts required for release, including the container images. Lastly, the engine deploys the latest changes to the necessary environment automatically, eliminating the need for manual deployment and monitoring [166, 189].

Monitoring Server (MS)

The Monitoring Server (MS), the telemetry server, supports the proactive management of system components by continuously analyzing the system metrics and logs [190, 166]. It tracks the system parameters from the IoT device, the edge server and the cloud subcomponents. Furthermore, it aggregates logs and metrics from these components and detects patterns for bugs, and provides insight into the overall status of the system.

The primary objective of the ML is to provide visibility into the system architecture's performance and an interface to track system metrics like CPU utilization, disk usage and network bandwidth, as required by FR-15 [17].

Central Experimental Tracking Server (CETS)

Like the experiment tracking server on the edge, the Central Experimental Tracking Server (CETS) tracks the ML training runs, running in the cloud infrastructure, to the persistent storage [19]. However, the CETS collects data regarding experiment runs from each edge server, providing a central view of all training runs across the hybrid edge-cloud system. Furthermore, the CETS provides an interface to compare the accuracy of machine learning models trained on the edge server against the universal models trained on the cloud [21].

Such comparisons provide insights to the CMLM and the system administrator to update the hyper-parameters for the training workflow for the cloud pipelines. Finally, the CETS is also responsible for tagging each experiment run with the edge server reference, enabling reproducibility and observability of the training process in the edge environment [191].

Central Model Registry (CMR)

For a hybrid ML architecture, where the edge server is responsible for training personalized models, tracking the ML models centrally for audit purposes is essential [161, 78]. Hence, the Central Model Registry (CMR) stores and logs all trained ML models centrally, enabling a centralized view of each model's deployment status in production, as required by FR-14.

The module is purpose-built to serve a dual function and to store the ML models trained on the cloud infrastructure and the edge servers [192]. In addition, the CMR also enables tagging the stages of the model, thereby allowing a model to transition from the staging phase to the production phase. This functionality provides a global system view and allows tracking of the deployed models in the hybrid edge-cloud environment.

Central Container Registry (CCR)

The primary pipeline in the CI/CD engine is to build a new container image based on the trigger by any code change. A Central Container Registry (CCR) is responsible for storing such container images for various edge and cloud modules [170]. This may include images for various ML systems, synchronization systems and edge and cloud orchestration engines.

Furthermore, the CCR can support access control policies, thereby only allowing authenticated edge servers to fetch the images, thus preventing unauthorized access to the code.

Central Storage

Central Storage in a cloud infrastructure refers to the primary data warehouse responsible for saving artifacts, public data, machine learning models, and experiment runs. The central storage provides the other components with a specific volume solution to run the independent processes, thereby enabling highly scalable system architecture, facilitating FR-8. Furthermore, central storage supports multiple storage tiers, primarily hot and cold. Hot storage refers to solutions that allow instantaneous data querying capability but at a higher cost. In contrast, cost storage provides a cheaper alternative for the long term, with a cost attached for each query for the data.

Central storage supports various life cycle policies, including transitioning from hot to cold storage after a specific interval. Such functionality optimizes the storage costs by archiving the stale and legacy public data and models to cold storage.

3.3 Behavioural View

A behavioural view of architecture describes the dynamic interactions between different system components and sub-components. The behavioural perspective explains the system's operation under different scenarios. Furthermore, the behavioural perspective describes the flow of control and messages from one component to another in various use cases, thereby outlining common workflows of the system.

In the behavioural view of the system, a sequence diagram demonstrates the interactions of how system components operate with one another to accomplish a particular task or workflow. To provide a holistic view, this work provides custom scenario diagrams that include structural and behavioural aspects. Such scenario diagrams represent the architecture and the flow of actions among its components. Furthermore, showing the structure and the behaviour together helps understand the role of each component in the system's functionality. This sub-section is dedicated to scenario diagrams for various workflows in a typical edge-cloud ML system.

3.3.1 Data Collection

Data serves as the backbone of any machine-learning system. The edge-cloud ML system can make informed predictions with accurate data, determining the overall application's

success. In the current architecture, there are two separate workflow activities for data collection.

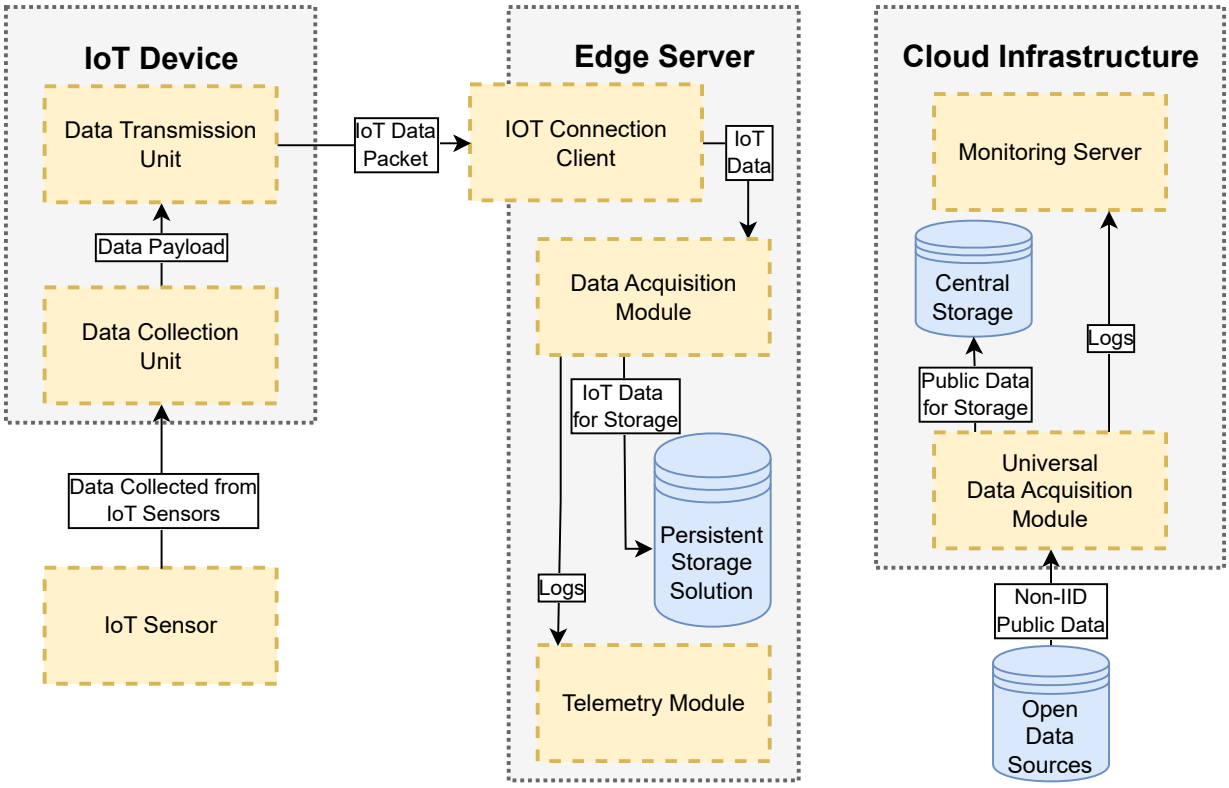


Figure 3.2: Data Collection Workflow

Firstly, the universal data collection in the cloud forms the primary activity sequence, which is responsible for gathering data for the global model, also known as the universal model. The Universal Data Acquisition Module (UDAM) interacts with the public data sources, extracts and processes the raw data, and stores the data in the Central Storage. As shown in Figure 3.2, the cloud data collection workflow gathers data and keeps the data ready for downstream ML tasks.

Secondly, the edge data collection workflow refers to the sequence of steps to collect data from the IoT devices in the edge environment. The IoT Connection Client (ICC) and the Data Acquisition Module (DAM) collaborate to accomplish the edge data collection, where the ICC provides necessary connection information about the IoT devices to the DAM. Furthermore, the DAM uses the connection information to extract data from IoT

devices. Depending on different applications, the DAM supports data scraping from the IoT device using pull methodology. In addition, the DAM supports streaming functionality, where the IoT device pushes data continuously as a stream, and the DAM acts as a stream consumer and saves data to persistent storage. Figure 3.2 explains the flow of data and commands in an edge environment for a data collection workflow.

3.3.2 Universal Training

Universal training refers to ML training in the cloud using publicly available data and the knowledge distilled from personalized edge models. The primary sub-components involved in this workflow include the Central Orchestration Engine (COE), Central ML Module (CMLM) and the CI/CD Engine. The COE triggers the CI/CD pipeline according to a scheduled cron job, where the CI/CD pipelines carry out this multi-step workflow, which includes multiple condition checks and data processes. Universal training is initiated when one of the following conditions meet:

- No universal model is found, based on querying the Central Model Registry (CMR).
- Multiple new personalized models are synchronized and perform better than the universal model. This information is extracted from the CMR and the Monitoring Server (MS).
- New data is available in the central storage.

The above checks are part of an initial condition verification step of the universal training process, where a successful response from any of the three conditions triggers the next step in the workflow. The next step includes preparing the data for training, where the data is queried from the central storage and processed according to the model requirements. Furthermore, based on the availability, the personalized models trained on the edge are fetched from the model registry and passed as input to the training step.

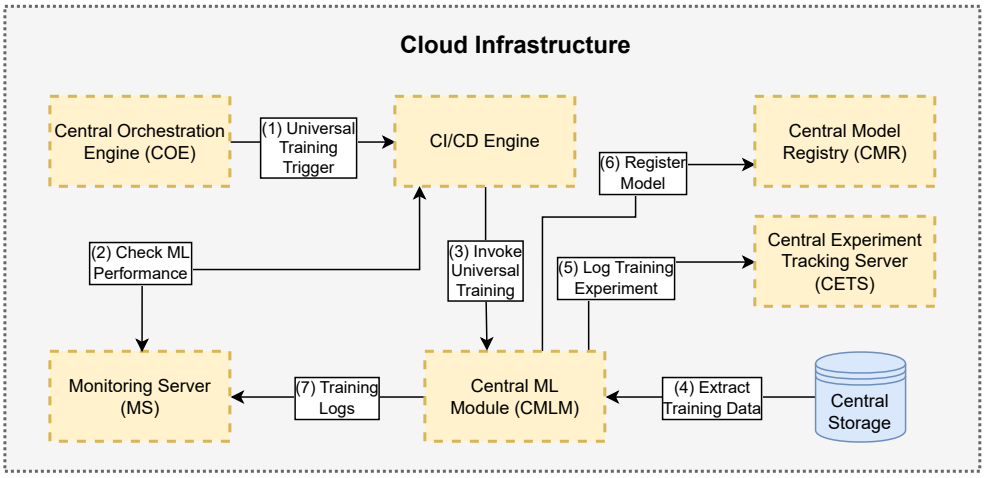


Figure 3.3: Universal Training Workflow

The universal training process, carried out by the CMLM, includes ML model training using the universal data fetched from the central storage and the extracted information from the personalized models. The final trained model is then saved to the CMR, while the training information is saved to the Central Experimental Tracking Server (CETS).

The CI/CD pipeline fetches the container images, also known as the code binaries, from the Central Container Registry (CCR) for each workflow step. Figure 3.3 demonstrates the sequence of steps for the universal training process.

3.3.3 Personalized Edge Training

Personalized Edge Training involves training ML models using IoT sensors' data collected near the edge. This step supports creating a model from scratch and retraining an old model using new data captured at the edge. Such a multi-step process is triggered by the Edge Orchestration Engine (EOE) based on scheduled time intervals. Firstly, the EOE queries the Telemetry Module (TM) for the current model's performance metrics. If the performance is below a predetermined threshold, the EOE triggers the next step in the workflow, as shown in Figure 3.4.

Secondly, the Edge ML Training Module (EMLTM) fetches the current production model from the Edge Model Registry (EMR), where the returned model could be an outdated personalized model or a universal model transferred from the cloud environment. Along with fetching the outdated model, the EMLTM extracts the personalized data from

the edge Persistent Storage Solution (PSS) and preprocesses the same for the model training process. Furthermore, the EMLTM retrain the model using the latest data and saves the newly trained model to the EMR. Finally, the EMLTM is also responsible for logging the training experiment to the Edge Experiment Tracking Server (EETS).

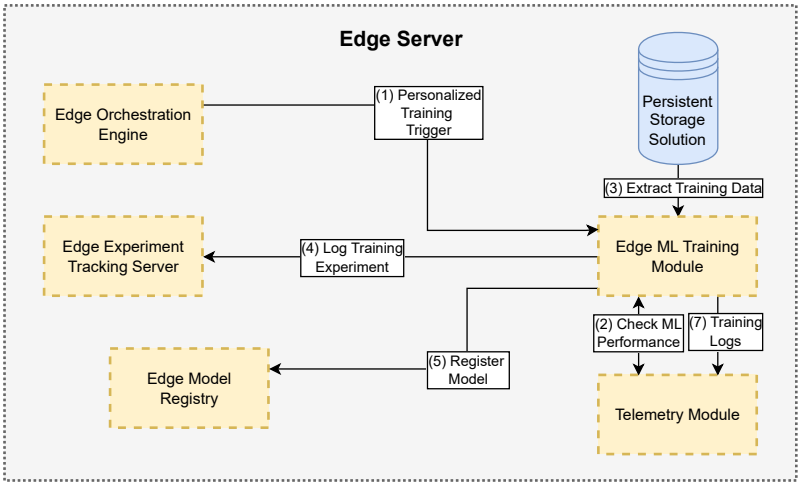


Figure 3.4: Personalized Edge Training Workflow

To run the edge training workflow, it is essential to preemptively fetch the training images from the Central Container Registry (CCR) to the Edge Container Registry (ECR). The CI/CD engine in the cloud infrastructure is responsible for fulfilling this prerequisite.

3.3.4 Inference

Prediction is the capability of an ML model to apply the learnings to unseen data. Model inference refers to utilizing a trained model for the prediction, using new data as input. The prerequisite of any inference process is fetching the latest model from the Edge Model Registry (EMR) and serving it for stream or batch prediction in the Edge ML Prediction Module (EMLPM).

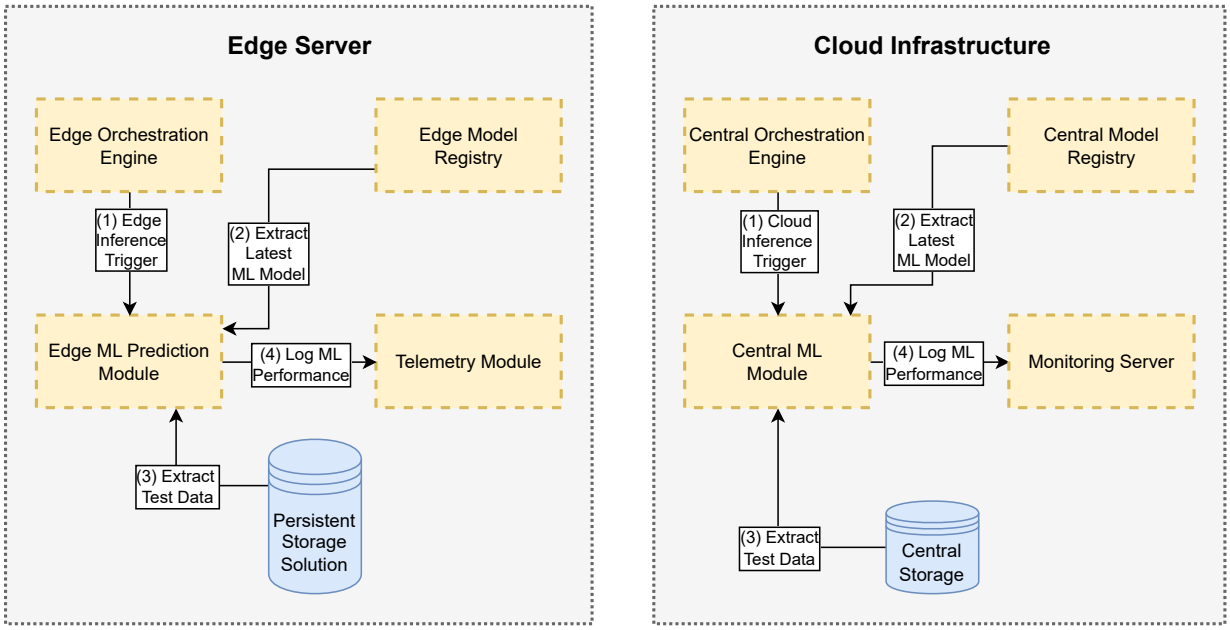


Figure 3.5: Inference Workflow

Based on the use case, the EMLPM accepts the raw data as input, uses the served model for prediction, and returns the predicted value. Furthermore, the EMLPM calculates the prediction performance of an ML model if the input data is labelled. Figure 3.5 demonstrates an ML system's inference and prediction sequences. Similarly, the Central ML Module (CMLM) generates prediction based on test data to evaluate the performance of the global model.

3.3.5 Continuous Synchronization

For observability and reproducibility, it is crucial for any distributed ML system that the personalized edge models and training runs are tracked globally. The Continuous Synchronization workflows include the experiment and model offloading from the edge to the cloud. Figure 3.6 illustrates the continuous model and experiment synchronization workflows.

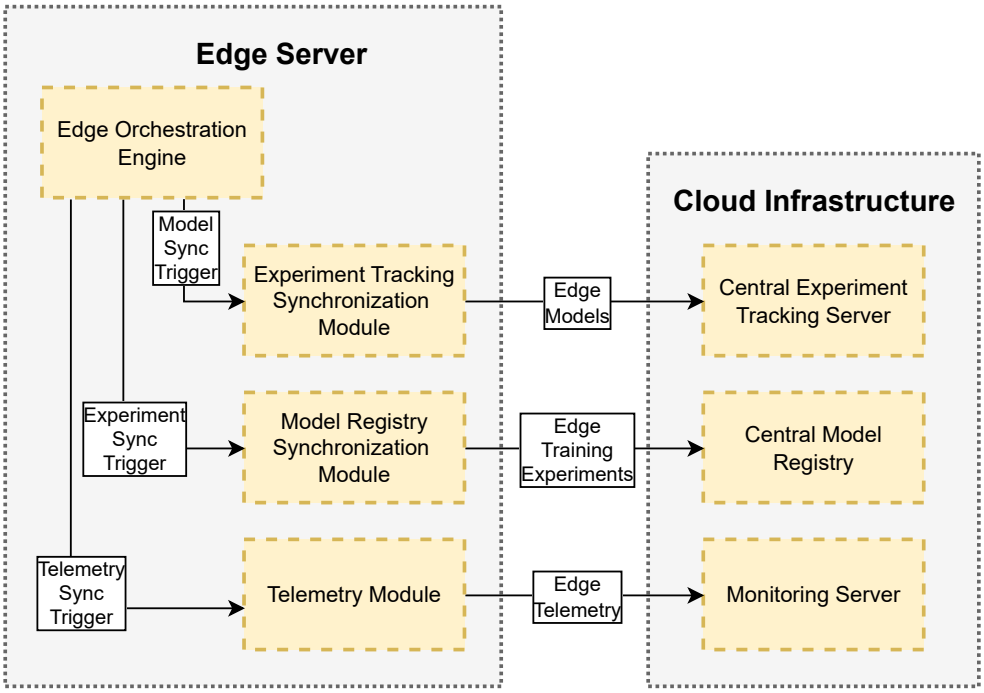
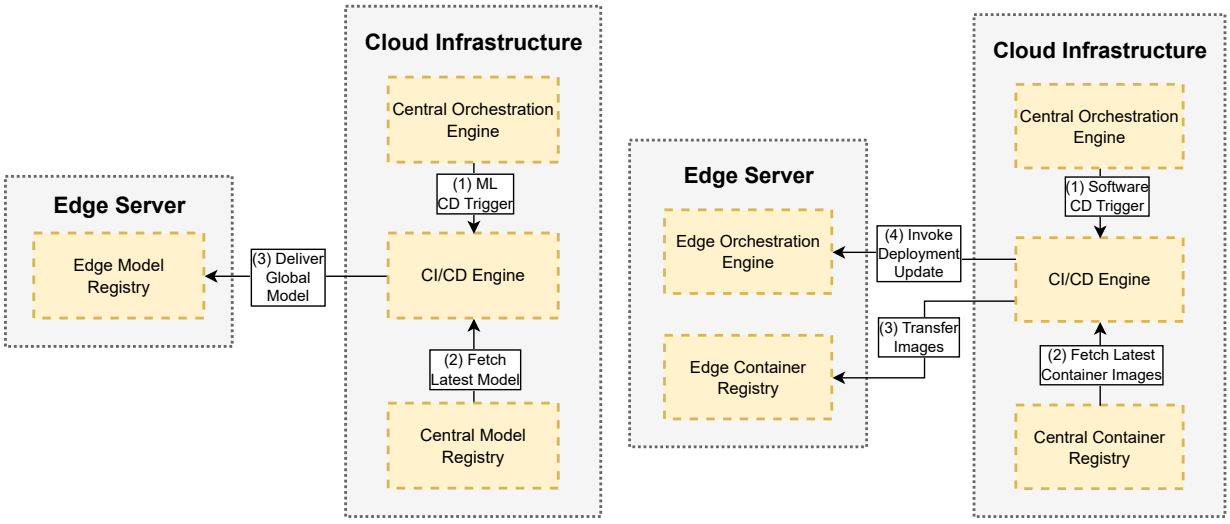


Figure 3.6: Continuous Synchronization Workflow

Based on the trigger from the Edge Orchestration Engine (EOE), the Model Registry Synchronization Module (MRSM) fetches the latest non-synchronized models from the Edge Model Registry (EMR) and uploads them to the Central Model Registry (CMR). Similarly, the Edge Experiment Tracking Server (EETS) offloads the edge training runs to the cloud to enable a system-wide view of ML training and ML performance.

3.3.6 Continuous Deployment

Continuous deployment refers to two different workflows in the context of an edge-cloud ML system. From a software perspective, continuous deployment includes deploying the latest code changes to the production environment. This workflow includes building the container images and updating them in the edge and the cloud. The CI/CD pipelines on the cloud coordinate the download of the latest container images on the edge container registry. Finally, the local containerization engine is responsible for replacing the running container by starting a new container using the latest image.



(a) CD Workflow for ML (b) CD Workflow for S/W

Figure 3.7: Continuous Deployment Workflows

From an ML respective, continuous deployment refers to the delivery and deployment of the universal ML model to the edge environment. Again, a particular CI/CD pipeline transfers the global model trained on the cloud to the Edge Model Registry (EMR). Figure 3.7 demonstrates the CD workflows for the software and machine learning processes.

Chapter 4

Software Architecture Analysis Method (SAAM)

In modern software development, evaluating a software architecture is pivotal in predicting the system's success. Section 3.1 defines the requirements of the system, while Section 3.2 provides the overview and structure of the components of the proposed architecture. Given the architecture's requirements and structure, it is essential to examine the alignment of the architecture with the underlying requirements [122].

Software Architecture Analysis Method (SAAM) is a consistent methodology to analyze and evaluate a software architecture's capability to cater to functional and non-functional requirements [193]. This chapter evaluates the architecture proposed in Chapter 3 using SAAM.

Kazman et al. proposed SAAM in 1994 for evaluating software architectures. SAAM undertakes an essential role in enabling the early detection and resolution of design issues in architecture. In addition, SAAM drives architectural refinement by comparing various architectural options. SAAM employs a scenario-based evaluation method which enables the representation of potential interactions of the system. This scenario-based method envisions various situations the system might face and how its architecture can handle them [194].

To evaluate an architecture, SAAM outlines a set of five activities [195]. This procedure starts with defining a standard way of dividing the functions and tasks of the system for a domain. It then involves mapping the functions onto the system's architectural components and modules. The next step in the procedure includes deciding on a selection of quality attributes that will be utilized for the assessment of the system. The last

two steps include choosing activities that test the quality attributes and evaluating if the system's components support completing each task. This aids in assessing the overall effectiveness of the architecture in fulfilling the selected quality attributes[195].

4.1 Motivation and Goals

In software engineering, architecture's functional and behavioural views are important in designing and implementing a software system. Developers use the functional view to divide the system into components and modules while clearly defining the functionality of each module. Similarly, the developers leverage the behavioural view to gain insights into the system's interactions and to understand the system's control flow. While such views provide a detailed understanding of the architecture, more is needed to determine the quality of the architecture conclusively.

To evaluate an architecture, quality attributes play a significant role by providing a new perspective of the system's performance beyond simple functionality [196, 96]. SAAM's scenario-based analysis approach considers such quality attributes to evaluate a software architecture. Understanding the impact of the architecture on quality attributes can provide insight into the system's performance in the real world, where quality attributes are equally important as the functionalities.

Evaluating a system architecture before implementation enables identifying design issues early in the development process, thereby preventing costly redesign efforts [197]. In addition, assessing how well the system's architecture handles various scenarios provides an iterative approach to fine-tuning and improving the architecture. For this purpose, this chapter uses SAAM to evaluate the proposed architecture and provides insights into various quality attributes of the architecture.

4.2 SAAM Analysis

In this section, the study follows the procedure outlined by Kazman et al., thus determining the quality attributes outlined in Step 3 of the SAAM [193]. Furthermore, as per Step 4 of SAAM, the scenarios for evaluating the architecture are selected. Finally, the component-scenario interactions are discussed to assess the effectiveness of the architecture in satisfying the quality attributes.

4.2.1 Quality Attributes for SAAM

In the context of selecting quality attributes, it is crucial to understand the system's domain. In the case of an edge-cloud ML hybrid ML system, the requirements originate from two domains: an ML domain and a software domain. Utilizing well-established standards to choose quality attributes for evaluating the architecture is a common practice. Thus, this study refers to two different standards for defining quality attributes.

A Machine Learning (ML) system is fundamentally a software system where the architecture still abides by the same practices of traditional system design. Hence, it is essential to approach an ML problem with the same discipline as any software system. ISO/IEC 25010:2011 [198] is a standard developed by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) that describes quality attributes for a software system. The standard divides the quality attributes into eight primary qualities: maintainability, reliability, functional suitability, performance efficiency, usability, security, compatibility, and portability [198, 195]. In addition, as per standard ISO/IEC 25059:2023 [199], AI systems require additional quality attributes for system evaluation, including user controllability, functional adaptability, correctness, intervenability and robustness.

This hybrid system selects a subset of attributes from the combination of ISO/IEC 25010:2011 and ISO/IEC 25059:2023. This study defines maintainability, reliability, scalability, functional adaptability, and robustness as the quality attributes to evaluate the system according to SAAM. Each quality attribute selected for evaluation is explained in brief below.

Maintainability

For a software system, maintainability refers to the property that determines the flexibility of a system to adapt to changes for error correction, system enhancements and accommodating changes [200]. While from an ML perspective, maintainability holds a similar definition but with some additional characteristics like the ability to update ML models. Hence, maintainability is a crucial attribute of an ML architecture.

Reliability

Reliability is a fundamental quality attribute in both ML and software systems. It refers to a system's capability to perform a specific function consistently over a defined period

without interruptions or failures [201]. Reliability can improve system performance and longevity in a distributed environment as an edge-cloud hybrid system.

Scalability

Scalability is the ability to adapt to increased workloads without significant degradation in performance by adding additional resources [202]. In the current system requirement, scalability can include adding new IoT sensors or increasing the number of ML models in the edge server. Furthermore, scalability could refer to adding edge servers to accommodate various use cases in multiple locations.

Functional Adaptability

Functional adaptability as a quality attribute of ML systems refers to the ability of an ML system to learn from the data from the past to make accurate predictions in the future. Furthermore, functional adaptability allows the system to continually improve and adapt based on new information, enabling better prediction performance.

Robustness

Robustness for an ML system is the ability to provide correct predictions in adversarial situations and be less vulnerable to errors and attacks [203]. This attribute is essential for an ML system's dependable operation in real-world scenarios. The robustness of an ML system primarily depends on the pre-processing workflows and the underlying ML models.

4.2.2 Scenarios

In SAAM, scenarios are defined as instances of the system's interactions and executions. Scenarios can also be represented as a sequence of actions between a system and its environment [204]. In this section, 18 scenarios are defined, along with short descriptions of their interactions. According to SAAM, a stakeholder is often included in an architecture's control sequence of a scenario. For an edge-cloud ML system, stakeholders can be characterized as data scientists, MLOps engineers, service managers and cloud engineers.

To manage the system, each stakeholder, also known as a user, bears duties about the architecture's usage. The data scientists are responsible for developing algorithms

for training ML models. The MLOps engineers manage the lifecycle of the ML models, including deployment and monitoring. Service managers manage the edge infrastructure involving the edge servers and the IoT devices. Finally, the cloud engineer is tasked with designing, deploying, and managing the organization's cloud system and resources. Various scenarios regarding each quality attribute are listed below, where each scenario is selected based on criticality, architectural impact, and frequency of occurrence in the real world.

Maintainability Scenarios

M-1 ***Transition Storage: Migrate storage to an external storage device on the edge server***

Due to the requirement for additional capacity, the service manager needs to change the data storage solution of the edge server to an external storage device.

M-2 ***Inspect Telemetry: View IoT device, Edge Server and Cloud telemetry data***

The service managers and the cloud engineers need a way to access, visualize and analyze telemetry data to monitor the system's health.

M-3 ***Transmit Control Messages: Send control signals to the IoT devices through the cloud interface***

The service manager may want to use the control interface hosted on the cloud to manage the IoT devices remotely. For example: enabling and disabling the transmission.

M-4 ***Accomodate New Model: Support a different type of ML model or ML training process***

The data scientist must update the code to train and use an ML model using the captured data, which must be deployed on the cloud or the edge server.

Reliability Scenarios

REL-1 ***Cloud System Recovery: Recover from a cloud infrastructure crash***

The edge server execution should continue in the case of a cloud infrastructure crash due to its distributed independent nature.

REL-2 ***Edge Server Fail-Safe: Use redundancy for edge server***

Multiple IoT devices upload data to the edge server, where the data is utilized for training and prediction. Adding a redundant edge server instance can minimize the possibility of data loss if the edge server experiences an breakdown or an outage.

REL-3 ***Cloud Fail-Safe: Use redundancy for cloud compute infrastructure***

The central management and control of the hybrid system should be able to recover in the case of cloud infrastructure failure, which can be achieved through additional cloud resources.

Scalability Scenarios

S-1 ***Offer Multiple Edge Models: Serve multiple ML models on the edge server***

To accommodate various applications, the Service manager and the MLOps engineer can decide to deploy multiple ML models on the edge server.

S-2 ***Facilitate Several Edge Servers: Support multiple edge servers at one location***

The service manager can deploy multiple edge servers at one tenant site to support more IoT devices with complex input like visual data.

S-3 ***Parallel Edge Training: Train multiple ML models in the edge server in parallel***

The MLOps engineer can support parallel training of ML models on the edge to reduce the training times.

S-4 ***Parallel Cloud Training: Train multiple ML models in the cloud***

The cloud and MLOps engineers can train multiple ML models in the cloud environment for various edge servers.

Functional Adaptability Scenarios

FA-1 ***Handle Faulty Data: Adapt to data drift due to faulty IoT device***

The ML model performance can degrade due to ingesting corrupted data from an IoT sensor. Recovering from such a scenario requires rollback capability.

FA-2 ***Mitigate Model Version Conflicts: Resolve conflict between new edge model and new global model***

The edge server would need to decide which ML model to select from the multiple newly trained models available on the edge server, which were personal and global training artifacts.

FA-3 ***Steady Performance Improvement: Improve accuracy with increasing data volume***

The performance of the ML models should improve with more time and data. The models should be capable of learning long-term patterns and provide better accuracy.

Robustness Scenarios

ROB-1 ***Manage Corrupted Data: Handle noisy data due to data corruption***

The ML model training and prediction pipeline should be able to handle anomalies in the input data.

ROB-2 ***Adversarial Robustness: Be resilient to adversarial attacks***

To ensure reliable performance, the ML model should be resilient to manipulated inputs as part of malicious attacks. The ML model should address model vulnerabilities during training and the inference phase.

ROB-3 ***Limited Resource Functionality: Operate in resource-constrained environments***

The model prediction, training performance, and IoT device management should be able to operate in a limited-resource infrastructure. For example, due to a faulty module on the edge server consuming a large amount of Random Access Memory(RAM), the overall working of the infrastructure should not be affected.

ROB-4 ***Disconnected Functionality: Operate in disconnected environments***

The connectivity and synchronization with the cloud should not affect the working of the edge server. Furthermore, the IoT devices should be able to recover from temporary disconnection with the edge server.

4.2.3 Evaluating Component-Scenario Interactions

According to Step 4 of SAAM, assessing if the architecture supports the scenarios is essential. The evaluation of scenarios helps to determine the architecture's applicability to support the scenario's use case. Thus, each scenario can be classified as direct or indirect based on the changes required in the architecture components. Hence, if a scenario operation can be directly mapped to a set of components, and the use case of the scenario can be achieved by various already supported architecture operations, the scenario can be characterized as direct. The indirect scenarios refer to the use cases requiring modifications to the current components to support the behaviour. The table evaluates the component-scenario interactions and classifies each scenario as direct or indirect. In addition, the table also provides the necessary changes to accommodate the indirect scenarios. Lastly, the table follows a similar format as presented in [195].

Table 4.1: Scenario Assessment

	Scenario	**Type of Change**	**Necessary Changes**	**Impacted Component(s)**
M-1	Transition Storage	Direct	-	-
M-2	Inspect Telemetry	Direct	-	-
M-3	Transmit Control Messages	Indirect	The Edge Server manages IoT devices. A change in the IoT Connection Client in the Edge Server can forward the control signals from the cloud to the IoT devices.	Edge Server - IoT Connection Client (ICC)
M-4	Accomodate New Mode	Direct	-	-
REL-1	Cloud System Recovery	Direct	-	-

Continuation on following page

Table 4.1 – content resumes from prior page

	Scenario	Type of Change	Necessary Changes	Impacted Component(s)
REL-2	Edge Server Fail-Safe	Indirect	The Edge server needs to be configured to work in Master-Backup environment, where the backup edge server can act as a master in the case of failure in the master node. Also, the IoT device must be aware of the backup edge server.	Edge Server - Edge Orchestration Engine (EOE). IoT device - Data Transmission Unit (DTU)
REL-3	Cloud Fail-Safe	Indirect	The cloud infrastructure needs to be configured with infrastructure as a Code(IaaC) with auto-scaling enabled. This solution enables the cloud to scale in case of higher demand or failure of a component	Cloud Infrastructure - CI/CD Engine
S-1	Offer Multiple Edge Models	Direct	-	-
S-2	Facilitate Several Edge Servers	Indirect	The cloud and the edge servers need to support multiple edge devices in the deployment location. Changes in the cloud can allow multiple edge servers to coordinate synchronization.	Cloud Infrastructure - COE
S-3	Parallel Edge Training	Indirect	Edge server cannot support parallel ML training due to resource constraints. Additional compute modules can be connected to the edge server for higher processing.	Edge Server - Physical Hardware, Edge ML Prediction Module (EMLPM)

Continuation on following page

Table 4.1 – content resumes from prior page

	Scenario	Type of Change	Necessary Changes	Impacted Component(s)
S-4	Parallel Cloud Training	Direct	-	-
FA-1	Handle Faulty Data	Indirect	Detecting erroneous data from the IoT devices requires the Edge server's DAM to validate data and add integrity checks.	Edge Server - Data Acquisition Module (DAM)
FA-2	Mitigate Model Version Conflicts	Direct	-	-
FA-3	Steady Performance Improvement	Direct	-	-
ROB-1	Manage Corrupted Data	Direct	-	-
ROB-2	Adversarial Robustness	Indirect	The EMLTM and CMLM should involve adversarial examples in the training process.	Edge Server - Edge ML Training Module (EMLTM). Cloud - Central ML Module (CMLM)
ROB-3	Limited Resource Functionality	Direct	-	-
ROB-4	Disconnected Functionality	Direct	-	-

Section 4.2.1 defines the quality attributes to evaluate the proposed edge-cloud ML system, while Section 4.2.2 describes various scenarios for each quality attribute to assess the behaviour of the architecture. According to SAAM, Table 4.1 classifies the nature of each scenario into direct and indirect according to the support needed to accommodate the requirements. Furthermore, the table highlights the necessary modifications required to the architecture to address the scenarios.

According to the scenario evaluation in Table 4.1, the required changes to the architecture are described for the indirect scenarios, along with the affected components of the

architecture. Furthermore, the sub-components, also known as the modules, require modifications to perform the behaviour listed in the scenarios. The below table presents the total number of changes required for each component based on the crafted scenarios.

Component	Total Number of Changes
IoT device	1
Edge Server	5
Cloud Infrastructure	3

Table 4.2: Number of Changes per Component

According to the data in Table 4.2, multiple components of the edge-cloud ML architecture need changes based on the scenarios. The IoT device requires only one change due to its independent nature based on the selected scenarios. Concurrently, the edge server requires multiple changes to accommodate the requirements, which reveals the complex nature of the operation of such a distributed system. Finally, the cloud infrastructure requires three changes according to the above assessment. A detailed analysis of each component against the quality attributes is provided below.

IoT device

The IoT device acts as the data source for the Edge-Cloud ML system, where various sub-components of the device are responsible for reliably transmitting data to the edge server. Furthermore, the IoT device can have multiple sensors to capture data from the environment, enabling multiple data streams for ML applications. Such support for parallel transmission of data from multiple streams facilitates the robust and efficient operation of the data acquisition pipeline.

Based on the scenarios described for SAAM, only one change is required for the IoT device. Furthermore, the performance of the IoT component against the specific set of quality measures is discussed below.

- **Maintainability** Due to the edge environment's distributed nature, managing multiple IoT devices is challenging. Hence, the maintainability of the IoT device in such

a constantly evolving environment is a significant quality attribute. In addition to transmitting sensor data, the IoT module also uploads telemetry data to the edge server. The crucial sub-component of the IoT device, the Data Transmission Unit (DTU), facilitates a bi-directional communication channel between the IoT device and the edge server. This communication channel provides an interface for the edge server to maintain the IoT device infrastructure. Such a management layer provides the maintainability attribute to the IoT device, allowing commands such as data transmission enable/disable to be transmitted according to the requirement. The telemetry data transmission enables the stakeholders to access the data according to ROB-4.

- **Reliability** Based on the scenarios, the IoT device is a reliable component due to its independent working. The two significant aspects to consider for the reliability of the IoT device are data integrity and the control message hierarchy. Data integrity refers to the trustworthiness of data collected from the IoT device. If the data sensor on the device is faulty or corrupted, the complete prediction pipeline can be affected. The Data Transmission Unit (DTU) of the IoT device and Edge ML Prediction Module (EMLPM) of the edge server safeguard against data corruption and altercation by processing data before using it for prediction.

  On the other hand, the control message hierarchy refers to the reliable operation of the control message transmission to and from the IoT device. Based on the scenario described by the REL-2, a minor modification is required to enable the IoT device to accommodate multiple edge servers. A failover mechanism switches to the redundant edge server, enabling a seamless transition and continuous data uploads to the edge infrastructure. The DTU should support multiple parent-edge servers for control and data transmission in this scenario.

  Based on the above discussion, the IoT device is a reliable component.

- **Scalability** The IoT device collects data from the onboard sensors and transmits the processed data to the edge server. Due to its more straightforward functionality, the modifications required due to the other scenarios do not impact the scalability of the IoT devices. Even with the scenario S-2, the support of multiple edge servers does not affect the functionality of the IoT device, as one IoT device only connects with one edge server at a time. Hence, increasing the number of IoT devices can be supported by either increasing data bandwidth between IoT devices and the edge server or allocating the new IoT devices to a separate edge server. The above direct change contributes towards making the architecture more scalable.

- **Functional adaptability** As shown in Table 4.1, the functional adaptability scenarios do not affect the IoT device directly. As this quality attribute is related to the machine learning functionality of the system, the IoT device does not directly affect the architecture's performance. In the case of FA-1, the data drift can occur due to the IoT device's fault sensor, but the solution to detect the drift involves changes on the edge server. Hence, the IoT device does not contribute negatively towards the functional adaptability attribute of the system.

- **Robustness** The robustness of an IoT sensor refers to its continuous, seamless and resilient operation in an edge environment. As described by scenario ROB-3, the operation of the IoT device in a resource-constrained environment contributes to the robustness of the component. Due to the nature of the IoT device's design, the device can operate in a low-powered computing environment. In addition, the efficient data processing onboard and power management techniques on an embedded device enable robust operation in an IoT environment. Hence, scenario ROB-3 does not require any change to the IoT device.

  According to the requirement of ROB-4, the IoT device should be capable of working in a disconnected environment. Again, due to the independent nature of the IoT device, the Data Transmission Unit (DTU) supports the operation of the device without connecting to the edge server. Furthermore, the DTU establishes a connection to the edge server when the connectivity is restored. The IoT sensor is a robust component based on the scenarios described above.

Edge server

As described in Section 3.2.2, the edge server handles various system tasks, including IoT device management, local data storage, training and prediction of ML models and synchronization with the cloud infrastructure. Hence, the edge server is the most crucial component of the multi-tier architecture. Table 4.1 analyses the scenarios and requires changes, often including the edge server. As described in the table, such changes are additive, enabling the seamless integration of the current architecture with the new and improved modifications. The evaluation of each quality attribute and the required changes is provided below.

- **Maintainability** Based on the initial requirements, the edge server is designed to be maintainable, using customizable and configurable components across the architecture. The IoT Connection Client (ICC) configures the connected devices, allowing

the edge server to control and manage the IoT infrastructure. In addition, using a container registry and a virtualization engine allows the use of portable and isolated system modules, deployable as containers on an edge server.

The edge server directly supports the requirements of scenario M-1 by using the virtualization engine on the edge server. The Persistent Storage Solution (PSS) can be directly configured to save data to an external device by mounting the new storage medium as a volume on the edge server. This functionality allows external data storage solutions to extend the data storage capacity of the edge server. The Telemetry Module (TM) on the edge server provides direct support to scenario M-2 by uploading logs and metrics to the cloud infrastructure, enabling a global view of the telemetry data.

According to scenario M-3, the edge server requires changes to support the message relay of control signals from the cloud to the IoT device. The primary solution to this requirement is to add a message relay service to the ICC. Such a service will forward authenticated control messages to the IoT device, allowing a global control panel deployed on the cloud to control the IoT infrastructure. As mentioned above, this new functionality can be an added change, which is deployed directly using the CI/CD component of the cloud infrastructure.

No immediate changes are required to the architecture to support the indirect scenario M-4. Such requirement is supported by the CD pipelines of the system, where the data scientist can push new code for training and use a new type of machine learning algorithm to the code repository. Once the changes by the data scientists are approved and merged, the CI/CD pipelines can automatically build the container images, deliver the images to the edge server and deploy the updated container in the edge environment without any manual intervention. Overall, these changes are additive changes and do not affect the system's maintainability. Hence, the edge server can be considered a maintainable component.

- **Reliability** As reliability is a significant factor in an ML system's performance, the edge server's proposed sub-components are designed with consistent operation as the primary objective. The edge server's virtualization engine enables the sub-component's reliable functioning by providing failover procedures. The engine recreates a new container in case of component failure. Such behaviour enables reliable operation in an edge environment.

  To accommodate the changes based on scenario REL-2, the edge server requires the Edge Orchestration Engine (EOE) to support a backup edge device. The EOE will maintain a flag to check if the current edge server acts as a master or a backup. Based

on this condition, the EOE can coordinate other processes on the edge server. Such a change can add significant reliability to the system and provide a failover process in case of a major component failure. The addition of such functionality can enhance the reliability of the system.

- **Scalability** The edge server acts as a bridge between the cloud infrastructure and the IoT devices. Hence it is primarily responsible for the scalability of the edge infrastructure. In addition, the edge server directly supports serving multiple ML models simultaneously, as per S-1. The Edge ML Prediction Module (EMLPM) can deploy various ML models using different endpoints on the edge server to support multiple applications.

  According to scenario S-3, two primary changes in the edge server are required to accommodate parallel training of ML models on the edge server. The first change refers to adding new computing resources to the system, which could include increasing the server's computing power using internal upgrades or attaching external compute modules. The second change requires updating the Edge ML Prediction Module (EMLPM) on the Edge server to support external computing environments. For example, the EMLPM can be configured to leverage external computing resources like GPUs, Field Programmable Gate Arrays (FPGAs) and Tensor Processing Units (TPUs) .

  The addition of such external computing modules can negatively contribute to the scalability of the system while expanding the overall capability of the system.

- **Functional adaptability** The edge server provides the primary ML capabilities to the proposed edge-cloud hybrid ML system. The edge server leverages the personalized data collected by the IoT sensors at the edge to train personalized models for future prediction. Any degradation in data from the IoT sensors can affect the performance of the edge server. According to scenario FA-1, the system's capability to adapt to the prediction degradation depends on the edge server. Hence, a modification in the Data Acquisition Module (DAM) is required to detect such degradation. The primary modification includes adding a pre-processing step in the DAM to validate the data and perform some data integrity checks. Such modification enables the system to detect and adapt to data drifts due to anomalies like faulty sensors and data corruption.

  The edge server directly supports the other scenarios related to functional adaptability, while FA-1 requires an additive change which can be integrated using the automated deployment pipelines. Hence, the edge server can be considered an adaptable component.

- **Robustness** The edge server acts as an independent component providing ML functionality in various system use cases. Hence, it is essential to include robustness as a quality attribute for assessment. The scenarios ROB-1, ROB-3 and ROB-4 do not require any changes to the edge server as the functionalities are supported directly by the architecture. In contrast, the scenario ROB-2 requires modifications to the edge server to become resilient against adversarial attacks. Adversarial attacks include modifications to input data to fool a ML model. Such attacks highlight the vulnerabilities in an ML process, and a malicious actor can exploit such vulnerabilities. To defend against such adversarial attacks, the training component needs to be updated in the edge server. The primary change includes adding adversarial examples to the Edge ML Training Module (EMLTM) training data. Such examples enable the training algorithm to detect adversarial examples and reduce the possibility of adversarial attacks. Hence, an addition of a new step in the training pipeline in the edge can significantly increase the robustness of the overall system.

Cloud Infrastructure

As proposed in Section 3.2.3, the cloud infrastructure monitors the overall operation of the system, ranging from the IoT devices to the Central ML Module (CMLM). The cloud infrastructure is also responsible for managing the edge servers, code integration, and deployment. Hence, the cloud can be considered a crucial central component of the architecture. Most of the required functionalities, as described by the scenarios, are fulfilled using the sub-components deployed in the cloud, while some scenarios require modification, as discussed in Table 4.1. As per SAAM, the effect of the cloud component on each quality attribute is assessed below.

- **Maintainability** As the cloud infrastructure is deployed on a virtual machine or as a distributed service managed centrally, the cloud provides better maintainability than the other components. Due to the nature of cloud computing, and the availability of resources in the cloud, the central architecture provides higher configuration options for computing, storage and networking resources. In addition to that, the central cloud interface can provide a global view of the IoT devices and the ML infrastructure.

  The scenario M-2 is directly supported by the architecture by providing a global view of the telemetry data using the Monitoring Server (MS). Similarly, as described in M-4, the updated logic and workflows can be automatically deployed using the CI/CD pipelines to accommodate a new machine learning model. Hence, there is no

need for any changes to the cloud architecture to add more maintainability as defined by the scenarios in Section 4.2.2.

- **Reliability** The cloud infrastructure includes various sub-modules and supports an external storage solution like object storage and SQL databases. Such external storage solutions provide in-build capabilities such as backups, snapshots and replication, which contribute to the system's overall reliability. Due to the use of CI/CD pipelines, the architecture can leverage the automated deployment capabilities of the system. Per scenario REL-1, a recovery procedure can be gracefully executed using CI/CD pipelines for cloud infrastructure failure. In addition, the backed-up data on the storage solutions support rollback capabilities, enabling a seamless transition to the recreated infrastructure.

  Furthermore, to enhance the reliability of the architecture, scenario REL-3 requires changes to the cloud infrastructure to support redundancy. The architecture can easily accommodate such functionality using cloud deployment features like auto-scaling and load-balancing. Each sub-component in the cloud architecture can be deployed independently with multiple compute instances for each service. Such compute instances can be abstracted as a single service using load balancers that balance the inter-component traffic across all compute nodes. Similar to the previous scenarios, such a change can be integrated seamlessly due to abundant resources in a cloud environment. Due to such capabilities, the cloud infrastructure can be classified as reliable.

- **Scalability** Scalability is the main benefit of the cloud computing paradigm, which can rapidly increase and decrease IT resources according to the application requirement. The cloud infrastructure provides flexible storage and compute resources using any cloud platform's horizontal and vertical scaling features. Due to the design of the Central ML Module (CMLM) in the cloud, the scenario S-4 is directly supported. As training an ML model is executed as a pipeline, it can be parallelized using multiple compute nodes in the cloud.

  In contrast, the scenario S-2 requires a minor change to the cloud infrastructure. The change is primarily confined to the Central Orchestration Engine (COE) in the cloud infrastructure. To support multiple edge servers to connect to the cloud from a similar location, the COE must be updated to allow synchronization with various edge servers. Again, such a change to the COE requires adding a new configuration feature that allows multiple edge servers. Hence, adding such minor features does not affect the system's overall performance and contributes positively to scalability.

- **Functional adaptability** The edge server and the cloud infrastructure define the functional adaptability of the overall system. As described in scenario FA-3, the overall accuracy of the system's prediction should increase with the advent of time and an increasing amount of data. Such behaviour is intrinsic in an ML system due to the nature of ML training algorithms. Hence, the functional adaptability of the cloud infrastructure is only dependent on the underlying ML model.

- **Robustness** The global ML training process on the cloud contributes to the overall ML robustness of the system. To defend against adversarial attacks, similar to the Edge ML Training Module (EMLTM), the Central ML Module (CMLM) needs to augment the training data with adversarial examples. Such an addition can enable the training process to train robust ML models, which are less vulnerable to such modifications. Such a change can address the challenge raised by scenario ROB-2.

  Owing to the characteristics of cloud computing, the ROB-3 and ROB-4 do not apply to the cloud infrastructure. Hence, the cloud infrastructure contributes highly to the robustness of the overall architecture.

4.3 Lessons Learned

The above analysis shows the architecture's performance using quality attributes according to Software Architecture Analysis Method (SAAM). Furthermore, various scenarios of the architecture's behaviour, including interactions between components and sub-components, are discussed to assess how well the architecture aligns with the system's requirements.

In terms of maintainability, the analysis reveals a need for a message relay in the edge server to enable the central management of IoT devices using the cloud infrastructure. The analysis also indicates the requirement for redundancy in the edge and the cloud environment, demonstrating the need for a fail-safe architecture. Furthermore, the investigation based on the scenarios uncovered the need for multiple edge servers in a location to scale to a significantly higher number of IoT devices. Lastly, the system performs well in terms of robustness and functional adaptability, with its dynamic training of ML models based on prediction performance and knowledge synchronization from the edge and the cloud.

The analysis revealed the need for external compute modules on the edge servers to improve the system's scalability for parallel ML training. Such addition of external computing components can negatively affect the reliability and maintainability of the system. Still, such limitation can be considered a trade-off to increase the architecture's scalability. Furthermore, the primary concern regarding the robustness against adversarial attacks can

be addressed by augmenting the training data with adversarial examples, enabling robust training. Such a change can be addressed by a data augmentation step in the training process on the edge and the cloud, which can be implemented using an added step in the training pipeline.

As discussed above, the changes required to address the scenarios, as per SAAM, are additive and can be accommodated in the module system with minor modifications. In summary, SAAM offered valuable guidance in identifying potential flaws in architectural design. In addition, SAAM proved to be a valuable tool in identifying solutions to address the architecture's shortcomings. As per analysis, the modular approach of the proposed architecture enables a relatively more straightforward integration of additive improvements and enhancements.

Chapter 5

Experiments

While theoretical architecture and simulations are crucial for the design of any system, real-world experimentation is also vital. An architecture can provide a general understanding of the behaviour of the system. However, real-world usage can help validate the assumptions and reveal the aspects of the environment that are not accurately modelled during the architecture design process [205, 206].

As proposed in Chapter 3, architecture is a theoretical representation of the system's blueprint without any real-world implementation. Furthermore, no practical existing implementation can be used as a benchmark to compare the analysis performed in Chapter 4. The Software Architecture Analysis Method (SAAM) provides valuable insights regarding the system's maintainability, scalability, reliability, functional adaptability and robustness. However, these insights continue to be theoretical and abstract without real-world reference.

The experiments carried out in this study represent an actual implementation of the architecture outlined in Chapter 3. These experiments allow us to test the theories and assumptions, thereby serving as a real-world representation of the architecture. Furthermore, such implementation also enables us to examine the applicability and validity of the analysis from Chapter 4.

5.1 Overview of Project

A practical application for a ML-enabled edge and cloud system is introduced to perform experiments and validate the architecture. The Distributed Machine Learning for Atmo-

spheric Conditions Evaluation in Cloud and Edge (DIMACE) is a technology that leverages the capabilities of edge and cloud computing to perform machine learning operations for weather forecasting applications. DIMACE stands for Distributed Machine Learning for Atmospheric Conditions Evaluation in Cloud and Edge.

The DIMACE project implements the proposed hybrid architecture with minor modifications and follows the MLOps lifecycle from data analysis to model deployment. The IoT devices collect weather data using onboard sensors while transmitting the processed data to the edge server. The edge server manages local data collection, personalized model training, model prediction and synchronization with the cloud. Finally, cloud infrastructure provides a global view of the system and trains universal ML models using distilled knowledge from the other edge models.

The project aims to demonstrate the application of the edge-cloud ML framework and provide a real-world and practical use case to evaluate the architecture against predefined qualitative and quantitative measures.

5.2 Setup

This experiment uses edge capabilities for ML-enabled real-time weather forecasting applications. The experiment was conducted in three different locations with unique atmospheric conditions and constant weather monitoring.

5.2.1 Hardware

The hardware setup for the DIMACE project comprises two primary components: the IoT device and the edge servers. Each edge server can support multiple IoT devices, enabling a single edge server setup per location. Such an approach provides higher scalability and lower latency for edge predictions.

Physical Devices

Due to its Wi-Fi capabilities, a Raspberry Pi Pico W is used as an IoT device to implement the current setup. The Raspberry Pi Pico W is a microcontroller board designed for embedded applications programmed in C, C++ or MicroPython. Furthermore, the Pico W has an Arm Cortex-M0+ processor, 264 KB SRAM and 2 MB flash, enabling cost-effective solutions for various IoT applications. It can collect sensor data, perform basic

processing and finally upload data using its wireless functionalities. In addition, the IoT device is connected to a BME280 sensor that measures various environmental parameters like temperature, humidity and atmospheric pressure. The IoT device interfaces with the BME280 sensor using the I2C protocol.

A Lenovo ThinkCentre M900 is selected for the edge servers due to its form factor and computing power. Equipped with an Intel(R) Core(TM) i7-6700T CPU @ 2.80GHz processor, 8 GB RAM and 256 GB onboard storage, the edge server acts as a perfect device for onboard ML computations and local data storage. Each location had one edge server and two IoT devices for the experiments.

Network

In the current setup, Wi-Fi technology is used for maintaining a stable connection between the IoT devices and the edge server. Furthermore, each IoT device has an onboard 2.4Ghz IEEE 802.11n [207] standard-compliant Wi-Fi module that enables a stable connection.

Using various communication protocols, the edge server can support an uplink connection to the cloud infrastructure. In the current setup, the edge server is connected to the internet using Gigabit Ethernet protocol, enabling a 1 Gbps data transfer rate. The edge server is also equipped with an IEEE 802.11n-compliant Wi-Fi chip [207], enabling communication between the edge server and the IoT device.

5.2.2 Software

A software system for a hybrid architecture involves different dependencies and frameworks according to the computing capability of the device or environment. For instance, an IoT device typically runs a lightweight operating system, while the cloud pipelines running ML workflows require more extensive dependencies and higher computing power.

IoT Environment

For the IoT device, MicroPython, version 1.20.0, is installed on the Raspberry Pi Pico W, along with an external library for the BME280 sensor for MicroPython. The MicroPython code deployed on the IoT device extracts weather data from the BME280 sensor and exposes the data over Wi-Fi protocol. The weather data is structured like a data dictionary and formatted as a JavaScript Object Notation (JSON) document. Finally, the Pico W

device hosts a lightweight web server that exposes multiple HTTP endpoints for extracting raw data and managing the IoT device.

Edge Environment

In this study, the edge server bridges the gap between the IoT devices and the cloud environment in a ML system. The edge servers demonstrate intelligent edge computing capabilities, enabling a global view of the IoT and ML deployments.

The edge server uses a docker environment [208] to host the various internal sub-components to create consistent and reproducible environments. The ability to encapsulate sub-component dependencies to a docker container enables isolation enhancing portability and security. Furthermore, containerizing each module into individual packages introduces less inter-dependency and encourages faster development cycles. Hence, each sub-component in an edge server is built and distributed as a docker image and finally deployed as a container.

Managing and running edge computing infrastructure can be challenging due to the architecture's distributed and ever-changing nature. Azure IoT Edge [209] is a managed service from the Microsoft Azure platform, which is discussed in more depth in Section 5.2.2. From the edge server's perspective, the Azure IoT edge manages the modules at the edge device, enabling a complete deployment cycle across multiple edge devices. Furthermore, the Azure IoT Edge supports various sub-components, also known as modules, as docker containers, leveraging the benefits of a docker environment. Finally, this managed service enables edge devices in offline environments where the connection to the cloud and the Azure platform is absent.

As discussed in Section 3, an edge server must support various storage solutions for an edge-cloud hybrid environment. Hence, to implement the proposed architecture, the experiment deploys a MySQL database server [210] and a Minio Object Storage [211] directly on the edge device as docker containers. The data acquisition process stores IoT data in the MySQL database, while the ML training and prediction module fetches raw data. Furthermore, a Minio Object Storage container is deployed to enable local object storage for ML models and training artifacts. In addition, the Minio instance also acts as MLflow's backend storage, enabling scalability and resilience for MLflow artifact storage. Finally, both the database and the storage solutions utilize docker volume mounts for storage, which can be pointed to an external storage device for further extensibility.

MLFlow, an open-source platform, is deployed as a container on the edge server to manage the complete lifecycle of the ML process. The MLFlow service tracks the ML experi-

ments and the trained models [78]. MLFLow provides a platform to log the training runs, including metrics, parameters, input data files and output artifacts. Furthermore, MLFlow supports model tracking through its model registry sub-component, enabling the edge system to track ML models efficiently. According to the proposed architecture, MLFlow acts as an experiment-tracking server and model registry for the edge environment.

Using Wi-Fi protocol, the Data Acquisition Module (DAM) on the edge server extracts data from the Raspberry Pi Pico W device. According to the network topology of the edge environment, the edge server and the IoT device are present on a similar Wi-Fi network, which enables wireless data extraction. In addition, as mentioned above, the DAM connects to the MySQL database to store the weather data locally.

For orchestrating the machine learning workflows, the backend and the Scheduler subcomponents work in unison to act as an Edge Orchestration Engine (EOE) in the edge environment. The Backend module and the Scheduler manage the ML and IoT device management workflows. Furthermore, the Backend module coordinates with the ML training and prediction modules to train, test and retrain the machine-learning model.

The Edge Machine Learning Module acts as a single module that takes on the responsibilities of the Edge ML Training Module (EMLTM) and Edge ML Prediction Module (EMLPM), thereby abstracting the ML functionality from the other modules. The Edge ML module runs the machine learning training, prediction and retraining workflows based on the triggers from the Backend module. Furthermore, the Dask library in the ML module enables the execution of parallel workflows and provides an endpoint to monitor the workflow progress. Finally, the last step of the ML module's training workflow interfaces with the MLFLow module to log the training run and the trained model.

According to the proposed architecture, the Model Registry Synchronization Module (MRSM) and Experiment Tracking Synchronization Module (ETSM) connect to the cloud infrastructure to perform two-way synchronization for trained models and experimental runs. Hence, in the current experiment, the MRSM and ETSM are responsible for uploading new models and experiments from the local MLflow to the MLFlow instance on the cloud. Furthermore, both modules are designed to wait for synchronization in case of network disconnection. Lastly, the MRSM is also responsible for fetching the latest universal ML model trained on the cloud and save in the local storage for future use.

The current experiment uses Logstash as a Telemetry Module (TM), which facilitates the log collection and log forwarding to the cloud. Similar to the other modules of the system, the Logstash container can continue functioning during network disconnections.

Cloud Environment

The cloud infrastructure provides the hybrid edge-cloud system's central management and monitoring interface. This experiment uses The cloud services as a Docker Swarm inside a virtual machine. Such infrastructure setup using Docker Swarm enables centralized management, which can be easily transitioned to distributed setup using managed services of a cloud platform.

Similar to the edge environment, the experiment uses MySQL, Minio Object storage and MLFlow for the underlying storage infrastructure and the machine learning experiment tracking. Furthermore, all these services currently deployed in a virtual machine can be scaled to more managed services like Microsoft Azure MySQL, Microsoft Azure Blob Storage and MLFlow by Azure Databricks [212].

The data acquisition module on the cloud acts as a Universal Data Acquisition Module (UDAM) and extracts data from OpenWeatherMap's weather data API, also referred to as OpenWeather API [213]. The module collects temperature, humidity and atmospheric pressure for a predefined location and stores it in the MySQL database.

Much like the edge infrastructure, the backend and the scheduler modules act as the triggers for the workflow pipelines for the data collection, model training, model evaluation and model retraining process. While the Central ML Module (CMLM) is responsible for training new machine learning models using universal data collected from public sources. Furthermore, the CMLM supports knowledge distillation from the personalized edge modules trained on the edge server to train a better-performing ML model for future iterations.

For log management and analytics, the cloud infrastructure uses the ELK stack, which includes ElasticSearch, Kibana and Logstash [214]. ElasticSearch is a NoSQL database that supports highly efficient text queries and stores massive amounts of logs. All the cloud modules forward their logs and metrics to Logstash, which pushes the logs to ElasticSearch. Furthermore, the Logstash module on the edge devices also pushes logs to the cloud Elasticsearch. Finally, the cloud infrastructure contains Kibana, a data visualization tool, to explore and visualize trends in errors, metrics and logs of the overall system.

This study maintains the CI/CD pipelines and the code using GitHub Actions and Code Repository [215]. Various manual and automated triggers across the repository initiate multiple workflows. Such workflows include testing the functionality of a module on creating a pull request and building and pushing the docker images for the updated modules on code merge to master. In addition, some examples of various workflows with manual triggers are cloud service deployment to the Azure Virtual Machine and IoT Edge Deployment using Azure IoT Hub for the edge servers.

5.3 Machine Learning Operations for DIMACE

In this section, the Machine Learning Operations (MLOps) is described as a sequence of steps for managing the experiment introduced in Section 5.1 . Such a process includes steps ranging from exploratory data analysis, feature extraction, model development, model evaluation and validation, model versioning, model deployment and model monitoring [216]. In the case of a hybrid system including edge and a cloud environment, the operations require more rigour and synchronization. This study provides a framework and architecture to develop and maintain a multi-tier system focusing on machine learning operations across environments.

The primary ML objective of the overall system is to predict future weather information using past knowledge and data. Hence, such a challenge can be categorized as a time-series forecasting problem. Overall, the problem formulation of DIMACE can be defined as:

> *Given the historical weather data, predict future data points for temperature, pressure and humidity.*

5.3.1 Dataset Exploration

This section focuses on the exploratory analysis of datasets collected on the edge and fetched from public data sources. The primary goal of this analysis is to understand the nature of the time-series dataset, including level, seasonality, trend, and noise. The data was collected on the edge using six IoT sensors from June 1st, 2023, to July 14th, 2023. Similarly, the data from public sources were extracted in the same date range.

The collected weather data consisted of three properties and metadata associated with the extraction process. The primary features of each data point include:

- timestamp - Time of Data Collection
- location - Location of Data Capture
- temperature - Temperature in Celsius
- humidity - Relative Humidity in Percentage
- pressure - Atmospheric Pressure in Hectopascal

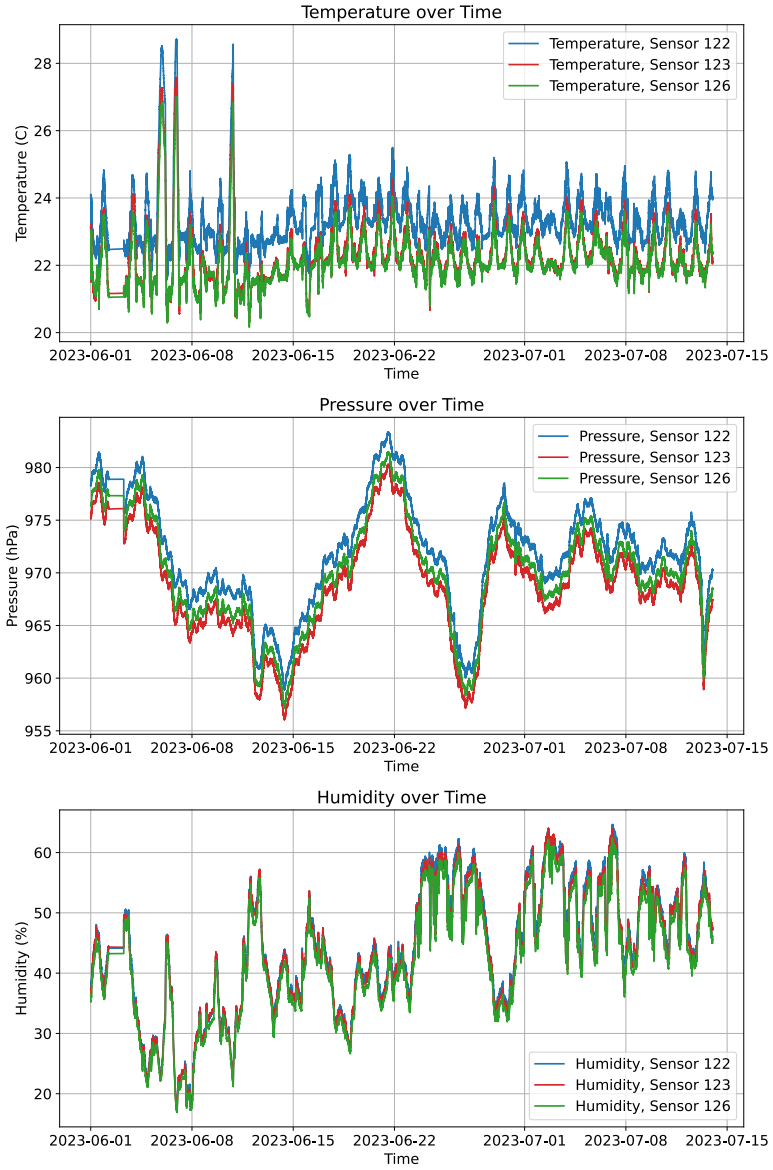


Figure 5.1: Time Series Progression of Data Collected on the Edge

Figure 5.1 and 5.2 illustrate the collected weather data from three IoT devices and public data collected from open sources. Figure 5.1 plots the temperature, pressure and humidity data collected from IoT devices labelled 122, 123 and 126, against time from June 1st, 2023, to July 14th, 2023. All three IoT devices were deployed around Building E5,

University of Waterloo, Waterloo, Canada. While, Figure 5.2 showcases the progression of time-series data for temperature, pressure and humidity extracted using public data sources for the Region of Waterloo, Canada.

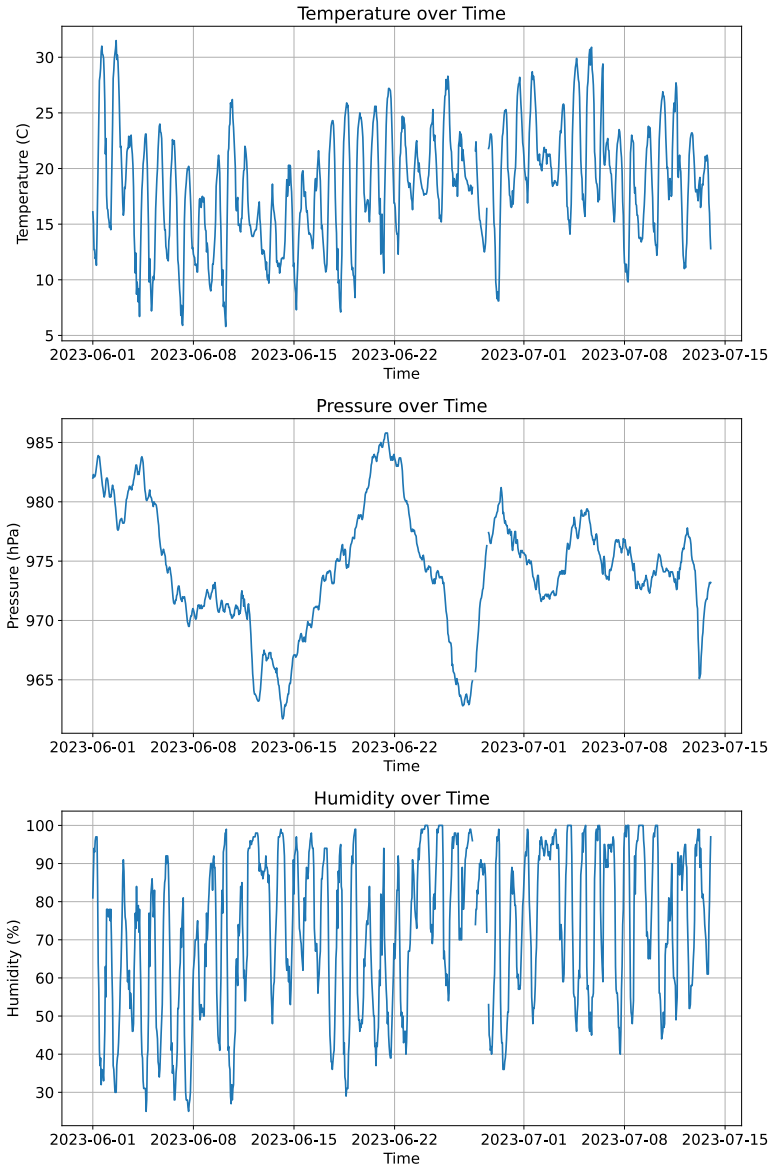
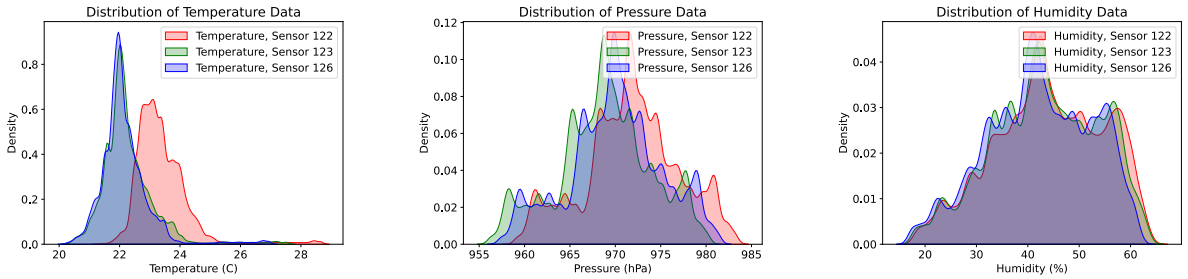


Figure 5.2: Time Series Progression of Data Collected from Public Sources

5.3.2 Data Analysis and Feature Engineering

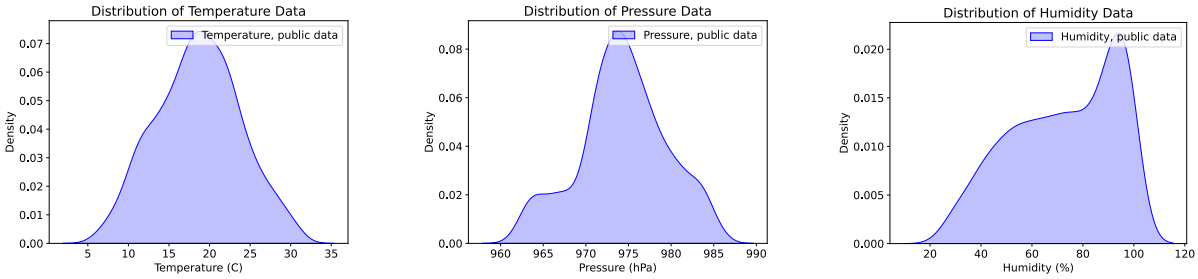
In the ML lifecycle, the data analysis step refers to the process of understanding the data and its underlying distribution. Figure 5.3 presents the Kernel Density Estimation (KDE) of various features of the weather data collected from IoT devices 122, 123 and 126. The figure reveals the drift in measurements across various devices, providing insights into the requirement for personalized models for each IoT device.



(a) Temperature (b) Pressure (c) Humidity

Figure 5.3: KDE plot for Data collected on the Edge

Similarly, Figure 5.4 displays the KDE for temperature, pressure and humidity for data collected on the cloud from OpenWeather API. Comparing the KDE of the public data with the KDE for the edge data, it is evident that there is a significant divergence in the data distribution across the two environments. Such a notable divergence advocates for personalized model training, enabling more accurate and context-aware predictions.



(a) Temperature (b) Pressure (c) Humidity

Figure 5.4: KDE Plot for Data collected from Public Sources

Furthermore, various methods for feature engineering were incorporated to derive meaningful features. For the case of a time series dataset, the standard approaches include calculating lag variables, moving averages and standard deviations. In the case of the DIMACE project, feature engineering is carried out by the underlying Autoregressive Integrated Moving Average (ARIMA) library, which involves capturing autoregressive and moving average terms, along with calculating the differencing component of the underlying data.

5.3.3 Model Training

In this experiment, based on the problem statement in Section 5.3 and the feature engineering in Section 5.3.2, there is a need for a statistical machine learning model. One such ML model for forecasting is the Autoregressive Integrated Moving Average (ARIMA) model, which captures various temporal relationships from time series data [217]. Furthermore, the ARIMA model divides the prediction task into three smaller sub-problems. First, the model uses a regression equation on the past data, also denoted as p. Secondly, the time series trend is extracted using the difference between the data and previous values, denoted as d. Finally, the model captures the moving average of the system, also referred to as q. Aggregating these components, the ARIMA model can be referred to as a parametric model with parameters p, d and q.

The ARIMA method provides a flexible approach to representing various time series structures with linear relationships. According to previous sections' data analysis and feature engineering, the ARIMA model would fit the current ML problem well. Using the past data for temperature, pressure and humidity data, a new ARIMA model is created for each atmospheric parameter using pre-calculated values for p, d and q.

5.3.4 Model Evaluation

The two primary steps to evaluate a machine learning model include selecting proper testing data and finalizing the error measure metrics. In the case of weather forecasting, the trained model can be evaluated by comparing forecasts to the data collected in the future. One common way to test future performance is to hold out a portion of the training data and test the model using the held-out portion.

Such techniques enable efficient testing of trained models and provide a performance score to predict a model's accuracy in production. For the case of this experiment, Mean

Squared Error (MSE) and Mean Absolute Error (MAE) are used as error metrics to measure how well the model fits the time-series data.

The primary goal of this thesis is to conceptualize and operationalize the MlOps architecture for a hybrid system. Therefore, the actual values of model predictions serve as secondary considerations and are not the focal point of this research.

5.3.5 Model Packaging

From an Machine Learning Operations (MLOps) perspective, the model packaging step is an essential aspect of the overall ML lifecycle, as it enables efficient storing and faster fetching of models in a production environment. For DIMACE, the ARIMA model is packaged as a pickle file, enabling efficient storage of the ML model in object storage. Furthermore, the pickle model is associated with an experiment using the experiment tracking server while also logging the model in the model registry.

Regarding input and output scalars for the model during inference, the ARIMA model uses the parameter *steps* that defines the future n intervals to predict where it outputs the value for n prediction in the future.

5.4 Quantitative Measurements

In an experiment, quantity attributes are essential to compare the experiment results as they provide a numerical basis to assess the overall system's performance. Furthermore, a quantitative approach to experimentation helps measure the reproducibility and consistency of an experiment. This section provides various quantitative measurements regarding the architecture's performance in a practical use case. It compares it to a generic ML architecture where the training is primarily conducted on the cloud. Such measurement provides a way to predict the performance of the proposed architecture in production environments. The generic cloud architecture is described in detail below.

To compare the proposed architecture against a generic cloud approach, a cloud-driven system is deployed as a separate Docker Stack. On the one hand, the newly built architecture can train on the cloud. On the other hand, the generic system has insufficient resources to train a new ML on the network edge. Hence, the generic cloud architecture continuously uploads the data to the cloud, and a new model is trained on the cloud and delivered to the edge. In the case of performance degradation of the model on the edge, a

workflow is triggered from the edge to initiate training on the cloud. When the training is complete, the cloud's CI/CD pipelines transfer the new model to the edge device. The proposed architecture's performance is compared against the generic cloud system using various quantitative measurements.

5.4.1 Response Time for Retraining

In any ML environment, achieving faster response time for retraining models in case of model degradation is crucial. Many edge applications require time-sensitive decision-making, where a faster retraining cycle implies faster updates to a model [218]. Such faster updates and deployment of better-performing models enable the efficient working of an edge application. Furthermore, in a rapidly changing edge environment similar to the weather prediction application, faster response time for retraining allows the model to adapt to the data drift and leads to better performance.

In the current investigation, we compare the response time of retraining using the proposed architecture with the conventional edge architecture that depends on the cloud for training a new machine learning model. In the case of DIMACE, a machine-learning model's average retraining and deployment time is 6.6 seconds, and the median time is 5.9 seconds. Such measurements are computed using 97 instances of model degradation during 44 days of operation.

	Average Training Time (s)	Average Deployment Time (s)	Average Response Time to Retrain (s)	Median Response Rime to Retrain (s)
Proposed Architecture	4.5	2.1	6.6	5.9
Cloud Architecture	3.6	19.0	22.6	20.3

Table 5.1: Response Time for Retraining and Deploying a Model

While for the case of machine learning training in the cloud, the time to retrain and deploy the newer model depends on the network bandwidth between the edge and the cloud. Furthermore, in this case, the edge server needs to synchronize the latest trained model

from the cloud and deploy it locally. According to the 60 instances of model deterioration, it took an average of 22.6 seconds to train the global model, transfer it to the edge and deploy the latest model on the edge server. At the same time, 20.3 seconds was the median time taken. It is important to note that the training time on the cloud was faster than the edge due to its higher computing capacity. Still, the overall time to recover from a data drift was significantly better in the case of an edge-cloud environment.

A statistical analysis is performed to determine if the difference between the performance measurements of both systems is significant. In this study, we perform a two-sample t-test based on the 97 data points for the proposed architecture and 60 data points for the generic cloud architecture. In this case, the null hypothesis defines no significant difference in the mean value for response time for retraining between both systems. The mean and the standard deviation of response to retraining time for the proposed architecture are 6.68 and 2.19 seconds, respectively. Similarly, 22.6 is the mean, and 5.73 is the standard deviation for the generic cloud system. Next, a p-value is calculated based on the distribution values mentioned above, which turns out to be less than 0.01. The null hypothesis is rejected because the significant level is below 5%, and the p-value is less than 0.05. Thus, the statistical analysis confirms that the proposed method performs significantly better than a generic cloud ML system in terms of responding to edge ML model degradation.

5.4.2 Bandwidth and Cost Comparison

The bandwidth and storage costs can be reduced compared to the conventional cloud computing approach by processing data closer to the data source. The proposed architecture significantly impacts the overall system's cost-effectiveness by leveraging the power of edge processing. Using the architecture's ability to train ML models on edge, there is a significant reduction in the data transfer from the edge to the cloud. Instead of sending raw unprocessed data to the cloud to train a new machine learning model, only the models, training experiment runs, and metrics are uploaded to the cloud, reducing the bandwidth costs of the system.

In the case of DIMACE, instead of sending 1.44 Gigabytes of data daily to the cloud for each IoT device, the new architecture only uploads the newly trained models and the experiments. As discussed in Section 5.4.1, the newly trained model and experiments were only synced for 97 instances of model degradation. The average bandwidth required for synchronizing model artifacts, training runs, and metrics is around 180 Megabytes per day for each IoT device. Such reduced bandwidth usage directly affects the overall cost of the system and can prove to be even more cost-efficient when the number of edge servers and

IoT scales to production scale. The theoretical bandwidth requirements of the proposed system and its comparison with the cloud ML system are provided in the table below.

	Edge-Cloud ML System (GB)	Cloud ML system (GB)
1 device	0.18 GB	1.44 GB
6 devices	1.08 GB	7.44 GB
100 devices	18 GB	144 GB

Table 5.2: Theoretical Bandwidth Comparison of Various Architectures

Furthermore, machine learning training on edge eliminates the need for massive central cloud storage to save the sensor data, as the data on edge can be discarded after training a model. Such optimizations can substantially decrease the overall running cost of the system. The savings from bandwidth costs, latency reduction and storage optimization can compensate for the upfront cost of deploying the edge servers.

5.4.3 Reliability

To quantify the reliability of the proposed architecture, several metrics of the DIMACE system are measured over a more extended period, including the development and deployment phases. The primary metrics associated with reliability include the system's operational time compared to the non-operational time. Such uptime is usually measured as the ratio of uptime and the total measurement time. Mean Time Between Failures (MTBF) is another system metric to assess the reliability of any system, where a higher MTBF is a property of a reliable system.

In the current experiments, the DIMACE system demonstrated an overall system uptime of 99.8%. Furthermore, the MTBF for IoT devices was around 3.5 days, while the MTBF for the edge server was recorded to be 11 days. Due to the robust nature of the cloud architecture, no direct failures were recorded during the experiment, except for minor disconnections at the time of updates to the cloud services.

5.5 Qualitative measurements

As discussed in Section 3.1.2, the Non-Functional Requirement (NFR) are crucial in a system's evaluation. Similar to the quantitative measurements, the qualitative attributes provide insights into the system's performance and alignment with user requirements. For evaluating the DIMACE system, privacy and security are the primary quality attributes to simulate the assessment of confidentiality and proper handling of sensitive user data.

5.5.1 Privacy

The primary benefit of processing the data closer to the source and training the machine learning models on the edge server is the privacy of the sensitive data generated by the IoT sensors. A privacy-centric ML approach is essential for various IoT applications deployed in locations like homes and personal vehicles. With the proposed architecture's ability to facilitate edge training while still offering a global system overview, privacy becomes a fundamental aspect in the design and implementation of the system.

In the DIMACE system, the weather data collected by the Raspberry Pi Pico W devices are stored temporarily on the edge server and used for local training of personalized models. Due to such functionality, the sensitive weather data is not uploaded to the cloud infrastructure, thus enhancing system privacy.

5.5.2 Security

Implementing machine learning training on the edge server significantly improves system security by reducing network data exposure and minimizing the attack surface for cyber exploits. Data exposure refers to the condition of a multi-tiered system where sensitive data is accessible to unauthorized actors by exposing various system vulnerabilities. In the context of an edge-cloud machine learning system, the data exposure can be significantly reduced by eliminating the need to transmit data over extended distances to the central cloud infrastructure.

Furthermore, distributing the computing workflows and persistent storage across various edge servers instead of centralized cloud infrastructure can minimize the impact of a single vulnerability a malicious actor can exploit. In the current experiment, the communication between the edge servers and the cloud infrastructure is limited to synchronizing training artifacts and metrics, thereby reducing the attack surface by eliminating raw data transmission over the internet.

5.6 SAAM vs Reality

According to SAAM, in Section 4, the proposed architecture has room for improvement in maintainability and scalability. However, the architecture's modular approach enables a relatively more straightforward addition of such improvements to the system. In terms of maintainability, similar to the architecture, on the one hand, DIMACE provided a monitoring server in terms of Kibana to access and visualize logs and metrics. On the other hand, DIMACE faced challenges in maintaining the IoT devices directly using a central interface installed on the cloud.

SAAM findings suggest the lack of redundancy in the edge environment and the cloud environment. While the overall system's operation resumed seamlessly in the case of cloud infrastructure's disconnection during updates, the same is not the case for the edge server. In the case of failure in the edge server, the data upload from the IoT devices stopped, causing a significant gap in the time series dataset. Hence, as deduced from the analysis from SAAM, the architecture can be improved by adding redundancy for the edge server and the cloud infrastructure.

In the context of scalability, SAAM provided suggestions to support multiple edge servers at a location and support external compute modules on the edge servers. Due to the reduced scale of the current experiment, the problems related to scalability did not manifest. Hence, the experiments could not be used to compare the experiment with SAAM in terms of scalability.

DIMACE exhibited optimal performance in terms of functional adaptability due to its model retraining functionality on the edge server itself. While DIMACE was not tested against adversarial attacks, as pointed out by SAAM, there is a significant need for adversarial model training using adversarial and perturbed examples. In the current use case, the ARIMA model is selected for time series forecasting, which can be vulnerable to adversarial attacks. Hence more advanced algorithms like the LSTM-ARIMA and the GRU-ARIMA can be used in the system that can benefit from adversarial training.

Similar to Software Architecture Analysis Method (SAAM), the real-world experiments indicate the need for improvements in the architecture in terms of maintainability, robustness and reliability. Furthermore, as demonstrated by DIMACE, the improvements discussed in this section and in Chapter 4 can improve the overall architecture.

Chapter 6

Conclusion

This thesis proposes a novel MLOps architecture to maintain machine learning applications in a hybrid edge-cloud environment. Initially, we define the functional and non-functional requirements for a hybrid edge-cloud ML system. Based on the formulated requirements, a functional view of multi-tier architecture is presented, including three main components: IoT device, edge server and cloud infrastructure. The main components of the architecture are described in brief below.

Firstly, the IoT device collects data from the sensors and transmits the data to the edge server. Secondly, the edge server handles various tasks ranging from data storage to machine learning model training. Furthermore, the edge server plays an essential role in the machine-learning workflows for personalized model training, including model training, model evaluation, model monitoring, experiment tracking and model tracking. Finally, the cloud infrastructure of the architecture acts as a central controller, ensuring seamless working of the edge server and the IoT devices. In addition, the cloud infrastructure is also responsible for training a global ML model using public data from public sources.

We further provide the behavioural view of the system to describe the interactions of the components and their sub-components under various scenarios. Such a view provides a perspective of how the system behaves over time. The primary scenarios described for the architecture include data collection, universal training, personal edge training, model inference, continuous synchronization and continuous deployment.

This thesis uses SAAM to assess the architecture and evaluate the system design based on pre-determined quality attributes. Based on various ISO standards, maintainability, scalability, reliability, functional adaptability and robustness are selected as quality attributes to evaluate the architecture. According to SAAM, various scenarios are defined,

which are used to assess individual components of the system.

SAAM revealed the lack of redundancy in the edge environment, negatively affecting the system's reliability. Furthermore, SAAM provided insights into the system's scalability by uncovering the need for multiple edge servers in a location for extending support to a high number of IoT sensors. Finally, regarding robustness, the investigation reveals the requirement of adversarial training to defend against adversarial attacks. However, further analysis revealed that the proposed architecture's modular approach enables the addition of these new changes into the system seamlessly.

To demonstrate the system's functionality and provide a real-world practical implementation of the architecture, various experiments are conducted on the DIMACE project. The experiments include monitoring a distributed weather forecasting system, which includes various IoT devices, edge servers and a cloud application. Furthermore, the proposed architecture is compared with a cloud architecture, where the qualitative and quantitative results from the experiments revealed better performance in terms of response time for model retraining, bandwidth usage and running cost.

While implementing the DIMACE project, similar deficiencies were encountered for various quality attributes, as indicated by SAAM, including scalability and reliability. Thus, the analytical approach of SAAM proved to be both practical and accurate. The improvements proposed in Chapter 4 establishes the foundation for future enhancements.

References

[1] Longbing Cao, Qiang Yang, and Philip S. Yu. Data science and AI in FinTech: an overview. *International Journal of Data Science and Analytics*, 12(2):81–99, August 2021. ISSN 2364-4168. doi: 10.1007/s41060-021-00278-w.

[2] Caiming Zhang and Yang Lu. Study on artificial intelligence: The state of the art and future prospects. *Journal of Industrial Information Integration*, 23:100224, September 2021. ISSN 2452-414X. doi: 10.1016/j.jii.2021.100224.

[3] Amisha, Paras Malik, Monika Pathania, and Vyas Kumar Rathaur. Overview of artificial intelligence in medicine. *Journal of Family Medicine and Primary Care*, 8 (7):2328–2331, July 2019. ISSN 2249-4863. doi: 10.4103/jfmpc.jfmpc_440_19.

[4] Jitendra Bhatia, Ridham Dave, Heta Bhayani, Sudeep Tanwar, and Anand Nayyar. SDN-based real-time urban traffic analysis in VANET environment. *Computer Communications*, 149:162–175, January 2020. ISSN 0140-3664. doi: 10.1016/j.comcom.2019.10.011.

[5] Tian-Xiang Sun, Xiang-Yang Liu, Xi-Peng Qiu, and Xuan-Jing Huang. Paradigm Shift in Natural Language Processing. *Machine Intelligence Research*, 19(3):169–183, June 2022. ISSN 2731-5398. doi: 10.1007/s11633-022-1331-6.

[6] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3):3713–3744, January 2023. ISSN 1573-7721. doi: 10.1007/s11042-022-13428-4.

[7] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchan. ChatGPT is not all you need. A State of the Art Review of large Generative AI models, January 2023. arXiv:2301.04655 [cs].

[8] Sukhpal Singh Gill, Minxian Xu, Carlo Ottaviani, Panos Patros, Rami Bahsoon, Arash Shaghaghi, Muhammed Golec, Vlado Stankovski, Huaming Wu, Ajith Abraham, Manmeet Singh, Harshit Mehta, Soumya K. Ghosh, Thar Baker, Ajith Kumar Parlikad, Hanan Lutfiyya, Salil S. Kanhere, Rizos Sakellariou, Schahram Dustdar, Omer Rana, Ivona Brandic, and Steve Uhlig. AI for next generation computing: Emerging trends and future directions. *Internet of Things*, 19:100514, August 2022. ISSN 2542-6605. doi: 10.1016/j.iot.2022.100514.

[9] Amanpreet Kaur Sandhu. Big data with cloud computing: Discussions and challenges. *Big Data Mining and Analytics*, 5(1):32–40, March 2022. ISSN 2096-0654. doi: 10.26599/BDMA.2021.9020016. Conference Name: Big Data Mining and Analytics.

[10] Saeedeh Parsaeefard, Iman Tabrizian, and Alberto Leon-Garcia. Artificial Intelligence as a Service (AI-aaS) on Software-Defined Infrastructure. In *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–7, October 2019. doi: 10.1109/CSCN.2019.8931372. ISSN: 2644-3252.

[11] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An Overview on Edge Computing Research. *IEEE Access*, 8:85714–85728, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2991734. Conference Name: IEEE Access.

[12] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, April 2022. ISSN 1319-1578. doi: 10.1016/j.jksuci.2021.11.019.

[13] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proceedings of the IEEE*, 107(8):1697–1716, August 2019. ISSN 1558-2256. doi: 10.1109/JPROC.2019.2915983. Conference Name: Proceedings of the IEEE.

[14] Asad Javed, Jérémy Robert, Keijo Heljanko, and Kary Främling. IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications. *Journal of Grid Computing*, 18(1):57–80, March 2020. ISSN 1572-9184. doi: 10.1007/s10723-019-09498-8.

[15] Yulei Wu. Cloud-Edge Orchestration for the Internet of Things: Architecture and AI-Powered Data Processing. *IEEE Internet of Things Journal*, 8(16):12792–12805, August 2021. ISSN 2327-4662. doi: 10.1109/JIOT.2020.3014845. Conference Name: IEEE Internet of Things Journal.

[16] Lanfang Sun, Xin Jiang, Huixia Ren, and Yi Guo. Edge-Cloud Computing and Artificial Intelligence in Internet of Medical Things: Architecture, Technology and Application. *IEEE Access*, 8:101079–101092, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2997831. Conference Name: IEEE Access.

[17] Sadman Sakib, Mostafa M. Fouda, Zubair Md. Fadlullah, and Nidal Nasser. Migrating Intelligence from Cloud to Ultra-Edge Smart IoT Sensor Based on Deep Learning: An Arrhythmia Monitoring Use-Case. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 595–600, June 2020. doi: 10.1109/IWCMC48107.2020.9148134. ISSN: 2376-6506.

[18] Luca Greco, Gennaro Percannella, Pierluigi Ritrovato, Francesco Tortorella, and Mario Vento. Trends in IoT based solutions for health care: Moving AI to the edge. *Pattern Recognition Letters*, 135:346–353, July 2020. ISSN 0167-8655. doi: 10.1016/j.patrec.2020.05.016.

[19] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. Towards MLOps: A Framework and Maturity Model. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 1–8, September 2021. doi: 10.1109/SEAA53835.2021.00050.

[20] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access*, 11:31866–31879, 2023. ISSN 2169-3536. doi: 10.1109/ACCESS.2023.3262138. Conference Name: IEEE Access.

[21] Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis, and George A. Papakostas. MLOps - Definitions, Tools and Challenges. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0453–0460, January 2022. doi: 10.1109/CCWC54503.2022.9720902.

[22] James Hutson, Kyle Coble, Naresh Kshetri, and Andrew Smith. Exploring the intersection of digital marketing and retail: Challenges and opportunities in ai, privacy, and customer experience. *Confronting Security and Privacy Challenges in Digital Marketing*, pages 50–72, 2023.

[23] Xiao Bai, Xiang Wang, Xianglong Liu, Qiang Liu, Jingkuan Song, Nicu Sebe, and Been Kim. Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments. *Pattern Recognition*, 120:108102, December 2021. ISSN 0031-3203. doi: 10.1016/j.patcog.2021.108102.

[24] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access*, 7:19143–19165, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2896880. Conference Name: IEEE Access.

[25] Éloi Zablocki, Hédi Ben-Younes, Patrick Pérez, and Matthieu Cord. Explainability of Deep Vision-Based Autonomous Driving Systems: Review and Challenges. *International Journal of Computer Vision*, 130(10):2425–2452, October 2022. ISSN 1573-1405. doi: 10.1007/s11263-022-01657-x.

[26] Jorge Ribeiro, Rui Lima, Tiago Eckhardt, and Sara Paiva. Robotic Process Automation and Artificial Intelligence in Industry 4.0 – A Literature review. *Procedia Computer Science*, 181:51–58, January 2021. ISSN 1877-0509. doi: 10.1016/j.procs.2021.01.104.

[27] Sébastien Lleo. Machine Learning: An Applied Mathematics Introduction. *Quantitative Finance*, 20(3):359–360, March 2020. ISSN 1469-7688. doi: 10.1080/14697688.2020.1725610. Publisher: Routledge _eprint: https://doi.org/10.1080/14697688.2020.1725610.

[28] Susmita Ray. A Quick Review of Machine Learning Algorithms. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 35–39, February 2019. doi: 10.1109/COMITCon.2019.8862451.

[29] Bin Wang, Jie Lu, Zheng Yan, Huaishao Luo, Tianrui Li, Yu Zheng, and Guangquan Zhang. Deep Uncertainty Quantification: A Machine Learning Approach for Weather Forecasting. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2087–2095, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330704.

[30] Yeongmin Ko, Younkwan Lee, Shoaib Azam, Farzeen Munir, Moongu Jeon, and Witold Pedrycz. Key Points Estimation and Point Instance Segmentation Approach for Lane Detection. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):8949–8958, July 2022. ISSN 1558-0016. doi: 10.1109/TITS.2021.3088488. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[31] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, July

2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3059968. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[32] Yujian Mo, Yan Wu, Xinneng Yang, Feilin Liu, and Yujun Liao. Review the state-of-the-art technologies of semantic segmentation based on deep learning. *Neurocomputing*, 493:626–646, July 2022. ISSN 0925-2312. doi: 10.1016/j.neucom.2022.01.005.

[33] Mahinda Mailagaha Kumbure, Christoph Lohrmann, Pasi Luukka, and Jari Porras. Machine learning techniques and data for stock market forecasting: A literature review. *Expert Systems with Applications*, 197:116659, July 2022. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.116659.

[34] Ramcharan Kakarla, Sundar Krishnan, and Sridhar Alla. Applied data science using pyspark.

[35] Pedro O O. Pinheiro, Amjad Almahairi, Ryan Benmalek, Florian Golemo, and Aaron C Courville. Unsupervised Learning of Dense Visual Representations. In *Advances in Neural Information Processing Systems*, volume 33, pages 4489–4500. Curran Associates, Inc., 2020.

[36] Michael Greenacre, Patrick J. F. Groenen, Trevor Hastie, Alfonso Iodice D'Enza, Angelos Markos, and Elena Tuzhilina. Principal component analysis. *Nature Reviews Methods Primers*, 2(1):1–21, December 2022. ISSN 2662-8449. doi: 10.1038/s43586-022-00184-w. Number: 1 Publisher: Nature Publishing Group.

[37] James B. Schreiber. Issues and recommendations for exploratory factor analysis and principal component analysis. *Research in Social and Administrative Pharmacy*, 17 (5):1004–1011, May 2021. ISSN 1551-7411. doi: 10.1016/j.sapharm.2020.07.027.

[38] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, March 2020.

[39] Quentin Fournier and Daniel Aloise. Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 211–214, June 2019. doi: 10.1109/AIKE.2019.00044.

[40] Kristina P. Sinaga and Miin-Shen Yang. Unsupervised K-Means Clustering Algorithm. *IEEE Access*, 8:80716–80727, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2988796. Conference Name: IEEE Access.

[41] Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide O. Oyelade, Laith Abualigah, Jeffery O. Agushaka, Christopher I. Eke, and Andronicus A. Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, April 2022. ISSN 0952-1976. doi: 10.1016/j.engappai.2022.104743.

[42] Algorithms for hierarchical clustering: an overview, II - Murtagh - 2017 - WIREs Data Mining and Knowledge Discovery - Wiley Online Library, .

[43] Vincent Cohen-addad, Varun Kanade, Frederik Mallmann-trenn, and Claire Mathieu. Hierarchical Clustering: Objective Functions and Algorithms. *Journal of the ACM*, 66(4):26:1–26:42, June 2019. ISSN 0004-5411. doi: 10.1145/3321386.

[44] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A Survey on Contrastive Self-Supervised Learning. *Technologies*, 9(1):2, March 2021. ISSN 2227-7080. doi: 10.3390/technologies9010002. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[45] Yassine Ouali, Céline Hudelot, and Myriam Tami. An Overview of Deep Semi-Supervised Learning, June 2020.

[46] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying Generalization in Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1282–1289. PMLR, May 2019. ISSN: 2640-3498.

[47] Brandon Butler. Cloud computing showdown: Amazon vs. Rackspace (OpenStack) vs. Microsoft vs. Google, December 2012.

[48] Teresa Guarda, Maria Fernanda Augusto, Ismael Costa, Pedro Oliveira, Datzania Villao, and Marcelo Leon. The Impact of Cloud Computing and Virtualization on Business. In Teresa Guarda, Filipe Portela, and Manuel Filipe Santos, editors, *Advanced Research in Technologies, Information, Innovation and Sustainability*, Communications in Computer and Information Science, pages 399–412, Cham, 2021. Springer International Publishing. ISBN 978-3-030-90241-4. doi: 10.1007/978-3-030-90241-4_31.

[49] Chnar Mustafa Mohammed and Subhi R. M. Zeebaree. Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review. *International Journal of Science and Business*, 5(2):17–30, 2021. Publisher: IJSAB International.

[50] Pallavi Wankhede, Minaiy Talati, and Rutuja Chinchamalatpure. Comparative study of cloud platforms -Microsoft Azure, Google Cloud Platform and Amazon EC2. *Journal of Research in Engineering and Applied Sciences*, 05(02):60–64, April 2020. ISSN 24566411, 24566403. doi: 10.46565/jreas.2020.v05i02.004.

[51] Arash Deldari and Alireza Salehan. A survey on preemptible IaaS cloud instances: challenges, issues, opportunities, and advantages. *Iran Journal of Computer Science*, 4(3):1–24, September 2021. ISSN 2520-8446. doi: 10.1007/s42044-020-00071-1.

[52] Salil Bharany, Kiranbir Kaur, Sumit Badotra, Shalli Rani, Kavita, Marcin Wozniak, Jana Shafi, and Muhammad Fazal Ijaz. Efficient Middleware for the Portability of PaaS Services Consuming Applications among Heterogeneous Clouds. *Sensors*, 22 (13):5013, January 2022. ISSN 1424-8220. doi: 10.3390/s22135013. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

[53] Subroto Roy and Nirmalya Bandyopadhyay. Implementing SaaS-Based Sales Force Automation Systems. *Indian Journal of Marketing*, 52(12):8–19, December 2022. ISSN 0973-8703. doi: 10.17010/ijom/2022/v52/i12/172559. Number: 12.

[54] Arokia Paul Rajan. A review on serverless architectures - function as a service (FaaS) in cloud computing. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(1):530–537, February 2020. ISSN 2302-9293. doi: 10.12928/telkomnika.v18i1.12169. Number: 1.

[55] Youssef Gahi and Imane El Alaoui. A Secure Multi-User Database-as-a-Service Approach for Cloud Computing Privacy. *Procedia Computer Science*, 160:811–818, January 2019. ISSN 1877-0509. doi: 10.1016/j.procs.2019.11.006.

[56] Sebastian Lins, Konstantin D Pandl, Heiner Teigeler, Scott Thiebes, Calvin Bayer, and Ali Sunyaev. Artificial intelligence as a service: classification and research directions. *Business & Information Systems Engineering*, 63:441–456, 2021.

[57] A. Shaji George, A. S. Hovan George, and A. S. Gabrio Martin. A Review of ChatGPT AI's Impact on Several Business Sectors. *Partners Universal International Innovation Journal (PUIIJ)*, 01(01):9–23, February 2023. doi: 10.5281/zenodo.7644359.

[58] Karansingh Chauhan, Shreena Jani, Dhrumin Thakkar, Riddham Dave, Jitendra Bhatia, Sudeep Tanwar, and Mohammad S. Obaidat. Automated Machine Learning: The New Wave of Machine Learning. In *2020 2nd International Conference on*

Innovative Mechanisms for Industry Applications (ICIMIA), pages 205–212, March 2020. doi: 10.1109/ICIMIA48430.2020.9074859.

[59] Dhananjay Singh, Gaurav Tripathi, Antonio M. Alberti, and Antonio Jara. Semantic edge computing and IoT architecture for military health services in battlefield. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 185–190, January 2017. doi: 10.1109/CCNC.2017.7983103. ISSN: 2331-9860.

[60] Satyendra K. Vishwakarma, Prashant Upadhyaya, Babita Kumari, and Arun Kumar Mishra. Smart Energy Efficient Home Automation System Using IoT. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–4, April 2019. doi: 10.1109/IoT-SIU.2019.8777607.

[61] IoT devices installed base worldwide 2015-2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, .

[62] The hidden cost of the edge | Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. https://dl.acm.org/doi/abs/10.1145/3458817.3476142, .

[63] Pasika Ranaweera, Anca Delia Jurcut, and Madhusanka Liyanage. Survey on Multi-Access Edge Computing Security and Privacy. *IEEE Communications Surveys & Tutorials*, 23(2):1078–1124, 2021. ISSN 1553-877X. doi: 10.1109/COMST.2021.3062546. Conference Name: IEEE Communications Surveys & Tutorials.

[64] Jiale Zhang, Bing Chen, Yanchao Zhao, Xiang Cheng, and Feng Hu. Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues. *IEEE Access*, 6:18209–18237, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2820162. Conference Name: IEEE Access.

[65] Dong-Meau Chang, Tse-Chuan Hsu, Chao-Tung Yang, and Junjie Yang. A Data Factor Study for Machine Learning on Heterogenous Edge Computing. *Applied Sciences*, 13(6):3405, January 2023. ISSN 2076-3417. doi: 10.3390/app13063405. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[66] Emmanuel Sibusiso Chaki, Sekgoari Semaka Mapunya, Emmanuel Sibusiso Chaki, and Mthulisi Velempini. Evaluating the Effectiveness of Machine Learning Algorithms in Detecting Distributed Denial of Service Attacks in Mobile Edge Computing. *International Conference on Intelligent and Innovative Computing Applications*, pages 264–269, December 2022. ISSN 1694-464X.

[67] Cristian Toma, Marius Popa, Bogdan Iancu, Mihai Doinea, Andreea Pascu, and Filip Ioan-Dutescu. Edge Machine Learning for the Automated Decision and Visual Computing of the Robots, IoT Embedded Devices or UAV-Drones. *Electronics*, 11(21): 3507, January 2022. ISSN 2079-9292. doi: 10.3390/electronics11213507. Number: 21 Publisher: Multidisciplinary Digital Publishing Institute.

[68] Madhavi Karanam, Krishna Chythanya Nagaraju, Gotham Sai P, SaiKiran Manasa S, and Pranay Krishna G. Granite classification using machine learning and edge computing. Technical Report 11:1276, F1000Research, November 2022. Type: article.

[69] Ganesh Ananthanarayanan, Yuanchao Shu, Landon Cox, and Victor Bahl. Project Rocket platform—designed for easy, customizable live video analytics—is open source, January 2020. Published: Microsoft Research Blog.

[70] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 159–173, October 2018. doi: 10.1109/SEC.2018.00019.

[71] Jacob Hochstetler, Rahul Padidela, Qi Chen, Qing Yang, and Song Fu. Embedded Deep Learning for Vehicular Edge Computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 341–343, October 2018. doi: 10.1109/SEC.2018.00038.

[72] Zhuangdi Xu, Harshit Gupta, and Umakishore Ramachandran. STTR: A System for Tracking All Vehicles All the Time At the Edge of the Network. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, DEBS '18, pages 124–135, New York, NY, USA, June 2018. Association for Computing Machinery. ISBN 978-1-4503-5782-1. doi: 10.1145/3210284.3210291.

[73] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, November 2016. arXiv:1602.07360 [cs].

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[75] Yue Zhou, Yue Yu, and Bo Ding. Towards MLOps: A Case Study of ML Pipeline Platform. In *2020 International Conference on Artificial Intelligence and Computer*

Engineering (ICAICE), pages 494–500, October 2020. doi: 10.1109/ICAICE51518.2020.00102.

[76] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen, and Tommi 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)*, pages 109–112, May 2021. doi: 10.1109/WAIN52551.2021.00024.

[77] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg, and Saransh 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)*, pages 25–28, December 2021. doi: 10.1109/AIKE52691.2021.00010.

[78] MLflow - A platform for the machine learning lifecycle. https://mlflow.org/, .

[79] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the Machine Learning Lifecycle with MLflow.

[80] Kubeflow ML workflows on Kubernetes. https://www.kubeflow.org/, .

[81] TFX | ML Production Pipelines. https://www.tensorflow.org/tfx, .

[82] Indika Kumara, Fabiano Pecorelli, Gemma Catolino, Rick Kazman, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. Architecting MLOps in the Cloud: From Theory to Practice. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 333–335, March 2023. doi: 10.1109/ICSA-C57050.2023.00076. ISSN: 2768-4288.

[83] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, and Fabio Antonelli. TinyMLOps: a framework for orchestrating ML applications at the far edge of IoT systems. In *2022 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–8, May 2022. doi: 10.1109/EAIS51927.2022.9787703. ISSN: 2473-4691.

[84] Stavros Kalapothas, Georgios Flamis, and Paris Kitsos. Efficient Edge-AI Application Deployment for FPGAs. *Information*, 13(6):279, June 2022. ISSN 2078-2489. doi: 10.3390/info13060279. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[85] Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing | IEEE Journals & Magazine | IEEE Xplore, .

[86] Jiasi Chen and Xukan Ran. Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, August 2019. ISSN 1558-2256. doi: 10.1109/JPROC.2019.2921977. Conference Name: Proceedings of the IEEE.

[87] Mário P Véstias, Rui Policarpo Duarte, José T de Sousa, and Horácio C Neto. Moving deep learning to the edge. *Algorithms*, 13(5):125, 2020.

[88] Zeyi Tao and Qun Li. {eSGD}: Communication efficient distributed deep learning on the edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[89] Emmanuel Raj, David Buffoni, Magnus Westerlund, and Kimmo Ahola. Edge MLOps: An Automation Framework for AIoT Applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–200, October 2021. doi: 10.1109/IC2E52221.2021.00034.

[90] Emmanuel Raj. Edge MLOps framework for AIoT applications, 2020. Accepted: 2020-06-04T06:46:34Z.

[91] Chulhong Min, Akhil Mathur, Utku Gunay Acer, Alessandro Montanari, and Fahim Kawsar. SensiX++: Bringing MLOPs and Multi-tenant Model Serving to Sensory Edge Devices, September 2021. arXiv:2109.03947 [cs].

[92] Shawn Hymel, Colby Banbury, Daniel Situnayake, Alex Elium, Carl Ward, Mat Kelcey, Mathijs Baaijens, Mateusz Majchrzycki, Jenny Plunkett, David Tischler, Alessandro Grande, Louis Moreau, Dmitry Maslov, Artie Beavis, Jan Jongboom, and Vijay Janapa Reddi. Edge Impulse: An MLOps Platform for Tiny Machine Learning, April 2023. arXiv:2212.03332 [cs].

[93] Raúl Miñón, Josu Diaz-de Arcaya, Ana I. Torre-Bastida, and Philipp Hartlieb. Pangea: An MLOps Tool for Automatically Generating Infrastructure and Deploying Analytic Pipelines in Edge, Fog and Cloud Layers. *Sensors*, 22(12):4425, January 2022. ISSN 1424-8220. doi: 10.3390/s22124425. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[94] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley Professional, 2000. ISBN 978-0-201-32571-3. Google-Books-ID: 3klAPCIB3hQC.

[95] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2003. ISBN 978-0-321-15495-8. Google-Books-ID: mdiIu8Kk1WMC.

[96] Mikael Svahnberg, Claes Wohlin, Lars Lundberg, and Michael Mattsson. A method for understanding quality attributes in software architecture structures. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, SEKE '02, pages 819–826, New York, NY, USA, July 2002. Association for Computing Machinery. ISBN 978-1-58113-556-5. doi: 10.1145/568760.568900.

[97] Lars Lundberg, Jan Bosch, Daniel Häggander, and Per-Olof Bengtsson. Quality Attributes in Software Architecture Design.

[98] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 41–50, September 2012. doi: 10.1109/RE.2012.6345838. ISSN: 2332-6441.

[99] Foundations for the study of software architecture | ACM SIGSOFT Software Engineering Notes, .

[100] Rick Kazman, Mario Barbacci, Mark Klein, S. Jeromy Carrière, and Steven G. Woods. Experience with performing architecture tradeoff analysis. In *Proceedings of the 21st international conference on Software engineering*, pages 54–63, Los Angeles California USA, May 1999. ACM. ISBN 978-1-58113-074-4. doi: 10.1145/302405.302452.

[101] Ruth Malan, Hewlett-Packard Company, Dana Bredemeyer, and Bredemeyer Consulting. Functional Requirements and Use Cases.

[102] Rongkai Wang, Chaojie Gu, Shibo He, Zhiguo Shi, and Wenchao Meng. An interoperable and flat Industrial Internet of Things architecture for low latency data collection in manufacturing systems. *Journal of Systems Architecture*, 129:102631, August 2022. ISSN 1383-7621. doi: 10.1016/j.sysarc.2022.102631.

[103] Jun-Song Fu, Yun Liu, Han-Chieh Chao, Bharat K. Bhargava, and Zhen-Jiang Zhang. Secure Data Storage and Searching for Industrial IoT by Integrating Fog Computing and Cloud Computing. *IEEE Transactions on Industrial Informatics*, 14(10):4519–4528, October 2018. ISSN 1941-0050. doi: 10.1109/TII.2018.2793350. Conference Name: IEEE Transactions on Industrial Informatics.

[104] David G Stork and Menlo Park. Open data collection for training intelligent software in the Open Mind Initiative.

[105] Isaac Odun-Ayo, Olasupo Ajayi, Boladele Akanle, and Ravin Ahuja. An Overview of Data Storage in Cloud Computing. In *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, pages 29–34, December 2017. doi: 10.1109/ICNGCIS.2017.9.

[106] Najmul Hassan, Saira Gillani, Ejaz Ahmed, Ibrar Yaqoob, and Muhammad Imran. The Role of Edge Computing in Internet of Things. *IEEE Communications Magazine*, 56(11):110–115, November 2018. ISSN 1558-1896. doi: 10.1109/MCOM.2018.1700906. Conference Name: IEEE Communications Magazine.

[107] Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and Weijia Jia. Fog-Based Computing and Storage Offloading for Data Synchronization in IoT. *IEEE Internet of Things Journal*, 6(3):4272–4282, June 2019. ISSN 2327-4662. doi: 10.1109/JIOT.2018.2875915. Conference Name: IEEE Internet of Things Journal.

[108] Rajesh Venkatesan, Madhava Venkatesh Raghavan, and K. Satya Sai Prakash. Architectural Considerations for a Centralized Global IoT Platform. In *2015 IEEE Region 10 Symposium*, pages 5–8, May 2015. doi: 10.1109/TENSYMP.2015.14.

[109] Troy Arcomano, Istvan Szunyogh, Jaideep Pathak, Alexander Wikner, Brian R. Hunt, and Edward Ott. A Machine Learning-Based Global Atmospheric Forecast Model. *Geophysical Research Letters*, 47(9): e2020GL087776, 2020. ISSN 1944-8007. doi: 10.1029/2020GL087776. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1029/2020GL087776.

[110] Rutvik V. Shah, Gillian Grennan, Mariam Zafar-Khan, Fahad Alim, Sujit Dey, Dhakshin Ramanathan, and Jyoti Mishra. Personalized machine learning of depressed mood using wearables. *Translational Psychiatry*, 11(1):1–18, June 2021. ISSN 2158-3188. doi: 10.1038/s41398-021-01445-0. Number: 1 Publisher: Nature Publishing Group.

[111] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. Machine Learning at Facebook: Understanding

Inference at the Edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344, February 2019. doi: 10.1109/HPCA.2019.00048. ISSN: 2378-203X.

[112] Panagiotis Kourouklidis, Dimitris Kolovos, Nicholas Matragkas, and Joost Noppen. Towards a low-code solution for monitoring machine learning model performance. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, pages 1–8, New York, NY, USA, October 2020. Association for Computing Machinery. ISBN 978-1-4503-8135-2. doi: 10.1145/3417990.3420196.

[113] Kai Yang, Tao Jiang, Yuanming Shi, and Zhi Ding. Federated Learning via Over-the-Air Computation. *IEEE Transactions on Wireless Communications*, 19(3):2022–2035, March 2020. ISSN 1558-2248. doi: 10.1109/TWC.2019.2961673. Conference Name: IEEE Transactions on Wireless Communications.

[114] Samuel Ackerman, Eitan Farchi, Orna Raz, Marcel Zalmanovici, and Parijat Dube. Detection of data drift and outliers affecting machine learning model performance over time, September 2022. arXiv:2012.09258 [cs, stat].

[115] Bo Yin, Hao Yin, Yulei Wu, and Zexun Jiang. FDC: A Secure Federated Deep Learning Mechanism for Data Collaborations in the Internet of Things. *IEEE Internet of Things Journal*, 7(7):6348–6359, July 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.2966778. Conference Name: IEEE Internet of Things Journal.

[116] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, June 2022. ISSN 0950-7051. doi: 10.1016/j.knosys.2022.108632.

[117] Federated Machine Learning: Concept and Applications: ACM Transactions on Intelligent Systems and Technology: Vol 10, No 2, .

[118] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On Non-Functional Requirements in Software Engineering. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. Yu, editors, *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*, Lecture Notes in Computer Science, pages 363–379. Springer, Berlin, Heidelberg, 2009. ISBN 978-3-642-02463-4. doi: 10.1007/978-3-642-02463-4_19.

[119] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer Science & Business Media, December 2012. ISBN 978-1-4615-5269-7. Google-Books-ID: MNrcBwAAQBAJ.

[120] Yongnan Zhang, Yonghua Zhou, Huapu Lu, and Hamido Fujita. Traffic Network Flow Prediction Using Parallel Training for Deep Convolutional Neural Networks on Spark Cloud. *IEEE Transactions on Industrial Informatics*, 16(12):7369–7380, December 2020. ISSN 1941-0050. doi: 10.1109/TII.2020.2976053. Conference Name: IEEE Transactions on Industrial Informatics.

[121] Ying-Feng Hsu, Ryo Irie, Shuuichirou Murata, and Morito Matsuoka. A Novel Automated Cloud Storage Tiering System through Hot-Cold Data Classification. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 492–499, July 2018. doi: 10.1109/CLOUD.2018.00069. ISSN: 2159-6190.

[122] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, July 2002. ISSN 1939-3520. doi: 10.1109/TSE.2002.1019479. Conference Name: IEEE Transactions on Software Engineering.

[123] Eric Samikwa, Antonio Di Maio, and Torsten Braun. Ares: Adaptive resource-aware split learning for internet of things. *Computer Networks*, 218:109380, 2022.

[124] Sensors | Free Full-Text | Edge Machine Learning for AI-Enabled IoT Devices: A Review, .

[125] Andreas P. Plageras, Kostas E. Psannis, Christos Stergiou, Haoxiang Wang, and B. B. Gupta. Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings. *Future Generation Computer Systems*, 82:349–357, May 2018. ISSN 0167-739X. doi: 10.1016/j.future.2017.09.082.

[126] Bhawna Gaur, Vinod Kumar Shukla, and Amit Verma. Strengthening People Analytics through Wearable IOT Device for Real-Time Data Collection. In *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, pages 555–560, April 2019. doi: 10.1109/ICACTM.2019.8776776.

[127] Yi-Ning Liu, Yan-Ping Wang, Xiao-Fen Wang, Zhe Xia, and Jing-Fang Xu. Privacy-preserving raw data collection without a trusted authority for IoT. *Computer Networks*, 148:340–348, January 2019. ISSN 1389-1286. doi: 10.1016/j.comnet.2018.11.028.

[128] Moataz Samir, Sanaa Sharafeddine, Chadi M. Assi, Tri Minh Nguyen, and Ali Ghrayeb. UAV Trajectory Planning for Data Collection from Time-Constrained IoT Devices. *IEEE Transactions on Wireless Communications*, 19(1):34–46, January 2020. ISSN 1558-2248. doi: 10.1109/TWC.2019.2940447. Conference Name: IEEE Transactions on Wireless Communications.

[129] Yuchen Li, Weifa Liang, Wenzheng Xu, and Xiaohua Jia. Data Collection of IoT Devices Using an Energy-Constrained UAV. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 644–653, May 2020. doi: 10.1109/IPDPS47924.2020.00072. ISSN: 1530-2075.

[130] Ievgeniia Kuzminykh, Anders Carlsson, Maryna Yevdokymenko, and Volodymyr Sokolov. Investigation of the IoT Device Lifetime with Secure Data Transmission. In Olga Galinina, Sergey Andreev, Sergey Balandin, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, Lecture Notes in Computer Science, pages 16–27, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30859-9. doi: 10.1007/978-3-030-30859-9_2.

[131] Ming-Shen Jian and Jimmy Ming-Tai Wu. Hybrid Internet of Things (IoT) data transmission security corresponding to device verification. *Journal of Ambient Intelligence and Humanized Computing*, March 2021. doi: 10.1007/s12652-021-03122-y.

[132] Pilar Andres-Maldonado, Pablo Ameigeiras, Jonathan Prados-Garzon, Juan J. Ramos-Munoz, and Juan M. Lopez-Soler. Optimized LTE data transmission procedures for IoT: Device side energy consumption analysis. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 540–545, May 2017. doi: 10.1109/ICCW.2017.7962714. ISSN: 2474-9133.

[133] Pilar Andres-Maldonado, Pablo Ameigeiras, Jonathan Prados-Garzon, Jorge Navarro-Ortiz, and Juan M. Lopez-Soler. Narrowband IoT Data Transmission Procedures for Massive Machine-Type Communications. *IEEE Network*, 31(6):8–15, November 2017. ISSN 1558-156X. doi: 10.1109/MNET.2017.1700081. Conference Name: IEEE Network.

[134] Abdul Aziz Chaudhry, Rafia Mumtaz, Syed Mohammad Hassan Zaidi, Muhammad Ali Tahir, and Syed Hassan Muzammil School. Internet of Things (IoT) and Machine Learning (ML) enabled Livestock Monitoring. In *2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*, pages 151–155, December 2020. doi: 10.1109/HONET50430.2020.9322666. ISSN: 1949-4106.

[135] Rupesh Bhandari and Kirubanand V B. Enhanced encryption technique for secure iot data transmission. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(5):3732, October 2019. ISSN 2722-2578, 2088-8708. doi: 10.11591/ijece.v9i5.pp3732-3738.

[136] Jianli Pan and James McElhannon. Future Edge Cloud and Edge Computing for Internet of Things Applications. *IEEE Internet of Things Journal*, 5(1):439–449, February 2018. ISSN 2327-4662. doi: 10.1109/JIOT.2017.2767608. Conference Name: IEEE Internet of Things Journal.

[137] Vahideh Hayyolalam, Moayad Aloqaily, Öznur Özkasap, and Mohsen Guizani. Edge-Assisted Solutions for IoT-Based Connected Healthcare Systems: A Literature Review. *IEEE Internet of Things Journal*, 9(12):9419–9443, June 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2021.3135200. Conference Name: IEEE Internet of Things Journal.

[138] Junxia Li, Jinjin Cai, Fazlullah Khan, Ateeq Ur Rehman, Venki Balasubramaniam, Jiangfeng Sun, and P. Venu. A Secured Framework for SDN-Based Edge Computing in IoT-Enabled Healthcare System. *IEEE Access*, 8:135479–135490, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3011503. Conference Name: IEEE Access.

[139] Gajanand Sharma, Naveen Hemrajani, Satyajeet Sharma, Aditya Upadhyay, Yogesh Bhardwaj, and Ashutosh Kumar. Data management framework for IoT edge-cloud architecture for resource-constrained IoT application. *Journal of Discrete Mathematical Sciences and Cryptography*, 25(4):1093–1103, May 2022. ISSN 0972-0529. doi: 10.1080/09720529.2022.2075086. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/09720529.2022.2075086.

[140] Muneeb Ejaz, Tanesh Kumar, Mika Ylianttila, and Erkki Harjula. Performance and Efficiency Optimization of Multi-layer IoT Edge Architecture. In *2020 2nd 6G Wireless Summit (6G SUMMIT)*, pages 1–5, March 2020. doi: 10.1109/6GSUMMIT49458.2020.9083896.

[141] Olivier Debauche, Saïd Mahmoudi, Sidi Ahmed Mahmoudi, Pierre Manneback, and Frédéric Lebeau. A new Edge Architecture for AI-IoT services deployment. *Procedia Computer Science*, 175:10–19, January 2020. ISSN 1877-0509. doi: 10.1016/j.procs.2020.07.006.

[142] Erkki Harjula, Pekka Karhula, Johirul Islam, Teemu Leppänen, Ahsan Manzoor, Madhusanka Liyanage, Jagmohan Chauhan, Tanesh Kumar, Ijaz Ahmad, and Mika

Ylianttila. Decentralized Iot Edge Nanoservice Architecture for Future Gadget-Free Computing. *IEEE Access*, 7:119856–119872, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2936714. Conference Name: IEEE Access.

[143] Junyong Wei, Jiarong Han, and Suzhi Cao. Satellite IoT Edge Intelligent Computing: A Research on Architecture. *Electronics*, 8(11):1247, November 2019. ISSN 2079-9292. doi: 10.3390/electronics8111247. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[144] Roger Young, Sheila Fallon, and Paul Jacob. An Architecture for Intelligent Data Processing on IoT Edge Devices. In *2017 UKSim-AMSS 19th International Conference on Computer Modelling & Simulation (UKSim)*, pages 227–232, April 2017. doi: 10.1109/UKSim.2017.19. ISSN: 2473-3520.

[145] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. Comparison of IoT platform architectures: A field study based on a reference architecture. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6, November 2016. doi: 10.1109/CIOT.2016.7872918.

[146] Shichao Chen, Qijie Li, Hua Zhang, Fenghua Zhu, Gang Xiong, and Ying Tang. An IoT Edge Computing System Architecture and its Application. In *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–7, October 2020. doi: 10.1109/ICNSC48988.2020.9238096.

[147] Meisei University, Tokyo, Japan, Tadashi Ogino, Shinji Kitagami, and Norio Shiratori. A Multi-agent Based Flexible IoT Edge Computing Architecture and Application to ITS. *Journal of Communications*, pages 47–52, 2019. ISSN 23744367. doi: 10.12720/jcm.14.1.47-52.

[148] Partha Pratim Ray and Neeraj Kumar. SDN/NFV architectures for edge-cloud oriented IoT: A systematic review. *Computer Communications*, 169:129–153, March 2021. ISSN 0140-3664. doi: 10.1016/j.comcom.2021.01.018.

[149] Navjot Kukreja, Alena Shilova, Olivier Beaumont, Jan Huckelheim, Nicola Ferrier, Paul Hovland, and Gerard Gorman. Training on the Edge: The why and the how. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 899–903, May 2019. doi: 10.1109/IPDPSW.2019.00148.

[150] Chen Chen, Bin Liu, Shaohua Wan, Peng Qiao, and Qingqi Pei. An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation

System. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1840–1852, March 2021. ISSN 1558-0016. doi: 10.1109/TITS.2020.3025687. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[151] Muhammad Babar, Mian Ahmad Jan, Xiangjian He, Muhammad Usman Tariq, Spyridon Mastorakis, and Ryan Alturki. An Optimized IoT-Enabled Big Data Analytics Architecture for Edge–Cloud Computing. *IEEE Internet of Things Journal*, 10(5):3995–4005, March 2023. ISSN 2327-4662. doi: 10.1109/JIOT.2022.3157552. Conference Name: IEEE Internet of Things Journal.

[152] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. AI on the Edge: Architectural Alternatives. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 21–28, August 2020. doi: 10.1109/SEAA51224.2020.00015.

[153] Kim-Hung Le, Khanh-Hoi Le-Minh, and Huy-Tan Thai. BrainyEdge: An AI-enabled framework for IoT edge computing. *ICT Express*, 9(2):211–221, April 2023. ISSN 2405-9595. doi: 10.1016/j.icte.2021.12.007.

[154] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar 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*, IoT '20, pages 1–8, New York, NY, USA, October 2020. Association for Computing Machinery. ISBN 978-1-4503-8758-3. doi: 10.1145/3410992.3411014. URL https://dl.acm.org/doi/10.1145/3410992.3411014.

[155] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1(4):216–222, April 2018. ISSN 2520-1131. doi: 10.1038/s41928-018-0059-3. Number: 4 Publisher: Nature Publishing Group.

[156] Tz-Heng Hsu, Zhi-Hao Wang, and Aaron Raymond See. A Cloud-Edge-Smart IoT Architecture for Speeding Up the Deployment of Neural Network Models with Transfer Learning Techniques. *Electronics*, 11(14):2255, January 2022. ISSN 2079-9292. doi: 10.3390/electronics11142255. Number: 14 Publisher: Multidisciplinary Digital Publishing Institute.

[157] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, and Sorin-Aurel Moraru. DevOps Transformation for Multi-Cloud IoT Applications. In *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, pages 1–6, August 2019. doi: 10.1109/ISSI47111.2019.9043730.

[158] Jitendcr Grover and Rama Murthy Garimella. Reliable and Fault-Tolerant IoT-Edge Architecture. In *2018 IEEE SENSORS*, pages 1–4, October 2018. doi: 10.1109/ICSENS.2018.8589624. ISSN: 2168-9229.

[159] Michael L. Hines, Thomas Morse, Michele Migliore, Nicholas T. Carnevale, and Gordon M. Shepherd. ModelDB: A Database to Support Computational Neuroscience. *Journal of Computational Neuroscience*, 17(1):7–11, July 2004. ISSN 1573-6873. doi: 10.1023/B:JCNS.0000023869.22017.2e.

[160] Samuel Idowu, Daniel Strüber, and Thorsten Berger. Asset Management in Machine Learning: State-of-research and State-of-practice. *ACM Computing Surveys*, 55(7): 144:1–144:35, December 2022. ISSN 0360-0300. doi: 10.1145/3543847.

[161] Christian Weber and Peter Reimann. MMP - A Platform to Manage Machine Learning Models in Industry 4.0 Environments. In *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 91–94, October 2020. doi: 10.1109/EDOCW49879.2020.00025. ISSN: 2325-6605.

[162] Gharib Gharibi, Vijay Walunj, Rakan Alanazi, Sirisha Rella, and Yugyung Lee. Automated Management of Deep Learning Experiments. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, DEEM'19, pages 1–4, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6797-4. doi: 10.1145/3329486.3329495.

[163] Samuel Idowu, Daniel Strüber, and Thorsten Berger. EMMM: A Unified Meta-Model for Tracking Machine Learning Experiments. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 48–55, August 2022. doi: 10.1109/SEAA56994.2022.00016.

[164] Jason Tsay, Todd Mummert, Norman Bobroff, Alan Braz, Peter Westerink, and Martin Hirzel. *Runway: machine learning model experiment management tool*. February 2018.

[165] Soumyalatha Naveen and Manjunath R Kounte. Key Technologies and challenges in IoT Edge Computing. In *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 61–65, December 2019. doi: 10.1109/I-SMAC47947.2019.9032541.

[166] Miguel A. López-Peña, Jessica Díaz, Jorge E. Pérez, and Héctor Humanes. DevOps for IoT Systems: Fast and Continuous Monitoring Feedback of System Availability.

IEEE Internet of Things Journal, 7(10):10695–10707, October 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.3012763. Conference Name: IEEE Internet of Things Journal.

[167] K.C. Gross, A. Urmanov, L.G. Votta, S. McMaster, and A. Porter. Towards Dependability in Everyday Software Using Software Telemetry. In *Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)*, pages 9–18, March 2006. doi: 10.1109/EASE.2006.21. ISSN: 2168-1872.

[168] Steve Buchanan, Janaka Rangama, and Ned Bellavance. Container Registries. In Steve Buchanan, Janaka Rangama, and Ned Bellavance, editors, *Introducing Azure Kubernetes Service : A Practical Guide to Container Orchestration*, pages 17–34. Apress, Berkeley, CA, 2020. ISBN 978-1-4842-5519-3. doi: 10.1007/978-1-4842-5519-3_2.

[169] João Pedro Dias, Hugo Sereno Ferreira, and Tiago Boldt Sousa. Testing and deployment patterns for the internet-of-things. In *Proceedings of the 24th European Conference on Pattern Languages of Programs*, EuroPLop '19, pages 1–8, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6206-1. doi: 10.1145/3361149.3361165.

[170] Roberto Morabito. Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. *IEEE Access*, 5:8835–8850, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2704444. Conference Name: IEEE Access.

[171] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network*, 32(1):102–111, January 2018. ISSN 1558-156X. doi: 10.1109/MNET.2018.1700175. Conference Name: IEEE Network.

[172] Wei Zhao, Jiajia Liu, Hongzhi Guo, and Takahiro Hara. ETC-IoT: Edge-Node-Assisted Transmitting for the Cloud-Centric Internet of Things. *IEEE Network*, 32 (3):101–107, May 2018. ISSN 1558-156X. doi: 10.1109/MNET.2018.1700164. Conference Name: IEEE Network.

[173] Heiko Koziolek, Sten Grüner, and Julius Rückert. A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In Anton Jansen, Ivano Malavolta, Henry Muccini, Ipek Ozkaya, and Olaf Zimmermann, editors, *Software Architecture*, Lecture Notes in Computer Science, pages 352–368, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58923-3. doi: 10.1007/978-3-030-58923-3_23.

[174] Anjus George. *Distributed Messaging System for the IoT Edge*. Ph.D., The University of North Carolina at Charlotte, United States – North Carolina, 2020. ISBN: 9798698560098.

[175] Claus Pahl, Nabil El Ioini, Sven Helmer, and Brian Lee. An architecture pattern for trusted orchestration in IoT edge clouds. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 63–70, April 2018. doi: 10.1109/FMEC.2018.8364046.

[176] Bill Wilder. *Cloud Architecture Patterns: Using Microsoft Azure*. "O'Reilly Media, Inc.", September 2012. ISBN 978-1-4493-5799-3.

[177] Y. Vishwanath, Raje Siddiraju Upendra, and Mohammed Riyaz Ahmed. A Review on Advent of IoT, Cloud, and Machine Learning in Agriculture. In Jennifer S. Raj, editor, *International Conference on Mobile Computing and Sustainable Informatics*, EAI/Springer Innovations in Communication and Computing, pages 595–603, Cham, 2021. Springer International Publishing. ISBN 978-3-030-49795-8. doi: 10.1007/978-3-030-49795-8_57.

[178] Yinglun Li, Weiliang Wen, Teng Miao, Sheng Wu, Zetao Yu, Xiaodong Wang, Xinyu Guo, and Chunjiang Zhao. Automatic organ-level point cloud segmentation of maize shoots by integrating high-throughput data acquisition and deep learning. *Computers and Electronics in Agriculture*, 193:106702, February 2022. ISSN 0168-1699. doi: 10.1016/j.compag.2022.106702.

[179] Tian Wang, Zhihan Cao, Shuo Wang, Jianhuang Wang, Lianyong Qi, Anfeng Liu, Mande Xie, and Xiaolong Li. Privacy-Enhanced Data Collection Based on Deep Learning for Internet of Vehicles. *IEEE Transactions on Industrial Informatics*, 16 (10):6663–6672, October 2020. ISSN 1941-0050. doi: 10.1109/TII.2019.2962844. Conference Name: IEEE Transactions on Industrial Informatics.

[180] Yuji Roh, Geon Heo, and Steven Euijong Whang. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, April 2021. ISSN 1558-2191. doi: 10.1109/TKDE.2019.2946162. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[181] Ke Xu, Yingguang Li, Changqing Liu, Xu Liu, Xiaozhong Hao, James Gao, and Paul G. Maropoulos. Advanced Data Collection and Analysis in Data-Driven Manufacturing Process. *Chinese Journal of Mechanical Engineering*, 33(1):43, May 2020. ISSN 2192-8258. doi: 10.1186/s10033-020-00459-x.

[182] Zi-hao Wang and Jing Wang. An IOT Data Collection Mechanism Based on Cloud-Edge Coordinated Deep Learning. In Songtao Guo, Kai Liu, Chao Chen, and Hongyu Huang, editors, *Wireless Sensor Networks*, Communications in Computer and Information Science, pages 76–89, Singapore, 2019. Springer. ISBN 9789811517853. doi: 10.1007/978-981-15-1785-3_6.

[183] Tongke Cui, Ruopeng Yang, Chao Fang, and Shui Yu. Deep Reinforcement Learning-Based Resource Allocation for Content Distribution in IoT-Edge-Cloud Computing Environments. *Symmetry*, 15(1):217, January 2023. ISSN 2073-8994. doi: 10.3390/sym15010217. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[184] Farzad Samie, Lars Bauer, and Jörg Henkel. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet of Things Journal*, 6(3):4921–4934, June 2019. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2893866. Conference Name: IEEE Internet of Things Journal.

[185] Ramón López-Viana, Jessica Díaz, and Jorge E. Pérez. Continuous Deployment in IoT Edge Computing : A GitOps implementation. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, June 2022. doi: 10.23919/CISTI54924.2022.9820108. ISSN: 2166-0727.

[186] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, and Sorin-Aurel Moraru. Continuous Testing in the Development of IoT Applications. In *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, pages 1–6, August 2019. doi: 10.1109/ISSI47111.2019.9043692.

[187] Charanjot Singh, Nikita Seth Gaba, Manjot Kaur, and Bhavleen Kaur. Comparison of Different CI/CD Tools Integrated with Cloud Platform. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 7–12, January 2019. doi: 10.1109/CONFLUENCE.2019.8776985.

[188] Badr El Khalyly, Abdessamad Belangour, Mouad Banane, and Allae Erraissi. A new metamodel approach of CI/CD applied to Internet of Things Ecosystem. In *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, pages 1–6, December 2020. doi: 10.1109/ICECOCS50124.2020.9314485.

[189] Ioannis Karamitsos, Saeed Albarhami, and Charalampos Apostolopoulos. Applying DevOps Practices of Continuous Automation for Machine Learning. *Information*,

11(7):363, July 2020. ISSN 2078-2489. doi: 10.3390/info11070363. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[190] Giuseppe Aceto, Alessio Botta, Walter de Donato, and Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, June 2013. ISSN 1389-1286. doi: 10.1016/j.comnet.2013.04.001.

[191] Julian Soh and Priyanshi Singh. Machine Learning Operations. In Julian Soh and Priyanshi Singh, editors, *Data Science Solutions on Azure: Tools and Techniques Using Databricks and MLOps*, pages 259–279. Apress, Berkeley, CA, 2020. ISBN 978-1-4842-6405-8. doi: 10.1007/978-1-4842-6405-8_8.

[192] Ask Berstad Kolltveit and Jingyue Li. Operationalizing machine learning models: a systematic literature review. In *Proceedings of the 1st Workshop on Software Engineering for Responsible AI*, SE4RAI '22, pages 1–8, New York, NY, USA, February 2023. Association for Computing Machinery. ISBN 978-1-4503-9319-5. doi: 10.1145/3526073.3527584.

[193] R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: a method for analyzing the properties of software architectures. In *Proceedings of 16th International Conference on Software Engineering*, pages 81–90, May 1994. doi: 10.1109/ICSE.1994.296768. ISSN: 0270-5257.

[194] SCAM: The Software Architecture Comparison Analysis Method. Technical report, . Section: Technical Reports.

[195] Goksen Umut Guler. Decentralized Data Acquisition Pipeline with Machine Learning For Side-Channel Information.

[196] Rick Kazman, Mark Klein, and Paul Clements. ATAM: Method for Architecture Evaluation:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, August 2000.

[197] Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM). Technical report, . Section: Technical Reports.

[198] Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. ISO ISO/IEC 25010:2011, Geneva, CH, 2011.

[199] Software engineering — systems and software quality requirements and evaluation (SQuaRE) — quality model for ai systems. ISO ISO/IEC 25059:2023, Geneva, CH, 2023.

[200] Rikard Land. Measurements of software maintainability. In *Proceedings of the 4th ARTES Graduate Student Conference*, pages 1–7. Citeseer, 2002.

[201] Michael R. Lyu. Software Reliability Engineering: A Roadmap. In *Future of Software Engineering (FOSE '07)*, pages 153–170, May 2007. doi: 10.1109/FOSE.2007.24.

[202] Gunnar Brataas and Peter Hughes. Exploring architectural scalability. In *Proceedings of the 4th international workshop on Software and performance*, WOSP '04, pages 125–129, New York, NY, USA, January 2004. Association for Computing Machinery. ISBN 978-1-58113-673-9. doi: 10.1145/974044.974064.

[203] Ronan Hamon, Henrik Junklewitz, and MARTIN Jose Ignacio Sanchez. Robustness and Explainability of Artificial Intelligence, January 2020. ISBN: 9789276146605 ISSN: 1831-9424.

[204] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, November 1996. ISSN 1937-4194. doi: 10.1109/52.542294. Conference Name: IEEE Software.

[205] James Bret Michael, Richard Riehle, and Man-Tak Shing. The verification and validation of software architecture for systems of systems. In *2009 IEEE International Conference on System of Systems Engineering (SoSE)*, pages 1–6, May 2009.

[206] Yujian Fu, Zhijiang Dong, and Xudong He. An approach to validation of software architecture model. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8 pp.–, December 2005. doi: 10.1109/APSEC.2005.33. ISSN: 1530-1362.

[207] IEEE 802.11n-2009. https://standards.ieee.org, .

[208] Docker: Accelerated, Containerized Application Development. https://www.docker.com/, May 2022.

[209] Azure IoT Edge documentation. https://learn.microsoft.com/en-us/azure/iot-edge/?view=iotedge-1.4.

[210] MySQL :: MySQL Documentation. https://dev.mysql.com/doc/, .

[211] MinIO Inc. MinIO | High Performance, Kubernetes Native Object Storage. https://min.io.

[212] Cloud Computing Services | Microsoft Azure. https://azure.microsoft.com/en-ca, .

[213] Weather API - OpenWeatherMap. https://openweathermap.org/api, .

[214] Elastic Stack: Elasticsearch, Kibana, Beats & Logstash. https://www.elastic.co/elastic-stack, .

[215] Features — GitHub Actions. https://github.com/features/actions, .

[216] Hamza Abubakar, M. S. Obaidat, Aaryan Gupta, Pronaya Bhattacharya, and Sudeep Tanwar. Interplay of machine learning and software engineering for quality estimations. In *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, pages 1–6, 2020. doi: 10.1109/CCCI49893.2020.9256507.

[217] William R. Kinney. ARIMA and Regression in Analytical Review: An Empirical Test. *The Accounting Review*, 53(1):48–60, 1978. ISSN 0001-4826. Publisher: American Accounting Association.

[218] Mollah Rezaul Alam, Kashem M. Muttaqi, and Abdesselam Bouzerdoum. Evaluating the effectiveness of a machine learning approach based on response time and reliability for islanding detection of distributed generation. *IET Renewable Power Generation*, 11(11):1392–1400, 2017. ISSN 1752-1424. doi: 10.1049/iet-rpg.2016.0987. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-rpg.2016.0987.

[219] Jianguo Chen, Kenli Li, Qingying Deng, Keqin Li, and Philip S. Yu. Distributed Deep Learning Model for Intelligent Video Surveillance Systems with Edge Computing. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2020. ISSN 1551-3203, 1941-0050. doi: 10.1109/TII.2019.2909473.

[220] Francisco-Javier Ferrández-Pastor, Higinio Mora, Antonio Jimeno-Morenilla, and Bruno Volckaert. Deployment of IoT Edge and Fog Computing Technologies to Develop Smart Building Services. *Sustainability*, 10(11):3832, November 2018. ISSN 2071-1050. doi: 10.3390/su10113832. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[221] Divyasheel Sharma and Santonu Sarkar. Enabling Inference and Training of Deep Learning Models for AI Applications on IoT Edge Devices. In Souvik Pal, Debashis De, and Rajkumar Buyya, editors, *Artificial Intelligence-based Internet of Things Systems*, Internet of Things, pages 267–283. Springer International Publishing, Cham, 2022. ISBN 978-3-030-87059-1. doi: 10.1007/978-3-030-87059-1_10.

[222] Anshul Ahuja, Geetesh Gupta, and Suman Kundu. A Serverless Approach to Federated Learning Infrastructure Oriented for IoT/Edge Data Sources (Student Abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):15747–15748, May 2021. ISSN 2374-3468. doi: 10.1609/aaai.v35i18.17870. Number: 18.

[223] Abdulrahman K. Alnaim and Ahmed M. Alwakeel. Machine-Learning-Based IoT–Edge Computing Healthcare Solutions. *Electronics*, 12(4):1027, January 2023. ISSN 2079-9292. doi: 10.3390/electronics12041027. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[224] Gawsalyan Sivapalan, Koushik Kumar Nundy, Soumyabrata Dev, Barry Cardiff, and Deepu John. ANNet: A Lightweight Neural Network for ECG Anomaly Detection in IoT Edge Sensors. *IEEE Transactions on Biomedical Circuits and Systems*, 16(1):24–35, February 2022. ISSN 1940-9990. doi: 10.1109/TBCAS.2021.3137646. Conference Name: IEEE Transactions on Biomedical Circuits and Systems.

[225] Mojtaba Eskandari, Zaffar Haider Janjua, Massimo Vecchio, and Fabio Antonelli. Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices. *IEEE Internet of Things Journal*, 7(8):6882–6897, August 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.2970501. Conference Name: IEEE Internet of Things Journal.

[226] Xingzhou Zhang, Yifan Wang, Sidi Lu, Liangkai Liu, Lanyu xu, and Weisong Shi. OpenEI: An Open Framework for Edge Intelligence. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1840–1851, July 2019. doi: 10.1109/ICDCS.2019.00182. ISSN: 2575-8411.

[227] Phu Nguyen, Nicolas Ferry, Gencer Erdogan, Hui Song, Stéphane Lavirotte, Jean-Yves Tigli, and Arnor Solberg. Advances in Deployment and Orchestration Approaches for IoT - A Systematic Review. In *2019 IEEE International Congress on Internet of Things (ICIOT)*, pages 53–60, July 2019. doi: 10.1109/ICIOT.2019.00021.

[228] Floriano De Rango, Giuseppe Potrino, Mauro Tropea, and Peppino Fazio. Energy-aware dynamic Internet of Things security system based on Elliptic Curve Cryptography and Message Queue Telemetry Transport protocol for mitigating Replay attacks.

Pervasive and Mobile Computing, 61:101105, January 2020. ISSN 1574-1192. doi: 10.1016/j.pmcj.2019.101105.

[229] Georgios Zachos, Ismael Essop, Georgios Mantas, Kyriakos Porfyrakis, José C. Ribeiro, and Jonathan Rodriguez. Generating IoT Edge Network Datasets based on the TON_iot Telemetry Dataset. In *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6, October 2021. doi: 10.1109/CAMAD52502.2021.9617799. ISSN: 2378-4873.

[230] Kolade Olorunnife, Kevin Lee, and Jonathan Kua. Automatic Failure Recovery for Container-Based IoT Edge Applications. *Electronics*, 10(23):3047, January 2021. ISSN 2079-9292. doi: 10.3390/electronics10233047. Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[231] Pekka Karhula, Jan Janak, and Henning Schulzrinne. Checkpointing and Migration of IoT Edge Functions. In *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '19, pages 60–65, New York, NY, USA, March 2019. Association for Computing Machinery. ISBN 978-1-4503-6275-7. doi: 10.1145/3301418.3313947.

[232] Pankaj Mendki. Docker container based analytics at IoT edge Video analytics usecase. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–4, February 2018. doi: 10.1109/IoT-SIU.2018.8519852.

[233] Omer Ali and Mohamad Khairi Ishak. Bringing intelligence to IoT Edge: Machine Learning based Smart City Image Classification using Microsoft Azure IoT and Custom Vision. *Journal of Physics: Conference Series*, 1529(4):042076, April 2020. ISSN 1742-6596. doi: 10.1088/1742-6596/1529/4/042076. Publisher: IOP Publishing.

[234] Yutaka Watanobe, Yuichi Yaguchi, Toshimune Miyaji, Ryuhei Yamada, and Keitaro Naruse. Data Acquisition Framework for Cloud Robotics. In *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, pages 1–7, October 2019. doi: 10.1109/ICAwST.2019.8923436. ISSN: 2325-5994.

[235] Xuan Qi and Chen Liu. Enabling Deep Learning on IoT Edge: Approaches and Evaluation. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 367–372, October 2018. doi: 10.1109/SEC.2018.00047.

[236] Zhu Tianqing, Wei Zhou, Dayong Ye, Zishuo Cheng, and Jin Li. Resource Allocation in IoT Edge Computing via Concurrent Federated Reinforcement Learning. *IEEE Internet of Things Journal*, 9(2):1414–1426, January 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2021.3086910. Conference Name: IEEE Internet of Things Journal.

[237] Arash Heidari, Nima Jafari Navimipour, Mohammad Ali Jabraeil Jamali, and Shahin Akbarpour. A green, secure, and deep intelligent method for dynamic IoT-edge-cloud offloading scenarios. *Sustainable Computing: Informatics and Systems*, 38:100859, April 2023. ISSN 2210-5379. doi: 10.1016/j.suscom.2023.100859.

[238] Ramón López-Viana, Jessica Díaz, Vicente Hernández Díaz, and José-Fernán Martínez. Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems. *IEEE Internet of Things Journal*, 7(10):10189–10199, October 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.3009633. Conference Name: IEEE Internet of Things Journal.