ARCADA

Edge MLOps framework for AIoT applications

Continuous delivery for AIoT, Big Data and 5G applications.

Emmanuel Raj

Master's Thesis

Master of Engineering - Big Data Analytics

June 3, 2020

MASTER'S THESIS	
Arcada University of Applied Sciences	
Degree Programme:	Master of Engineering - Big Data Analytics
Identification number:	7751
Author:	Emmanuel Raj
Title:	Edge MLOps framework for AIoT applications Continuous delivery for AIoT, Big Data and 5G applications.
Supervisor (Arcada):	Magnus Westerlund
Commissioned by:	TietoEvry

Abstract:
Recent years witnessed a boom in IoT devices resulting in big data and demand for low latency communication giving rise to a demand for 5G Networks. This shift in the infrastructure is enabling real-time decision making using artificial intelligence for IoT applications. Artificial Intelligence of Things (AIoT) is the combination of artificial intelligence (AI) technologies with the Internet of Things (IoT) infrastructure to achieve more efficient IoT operations and decision making. Edge computing is emerging to enable AIoT applications. Edge computing enables generating insights and making decisions at the data source, reducing the amount of data sent to the cloud and central repository. An ecosystem to facilitate edge computing for AIoT applications has become essential to make real-time decisions at the data source. In this thesis, we develop a framework to facilitate machine learning at the edge for AIoT applications which enables continuous delivery, deployment and monitoring of Machine Learning models at the edge (Edge MLOps). We will propose an ideal architecture, services, tools and methods for optimization of costs, operations, and resources to facilitate efficient edge-cloud operations at scale using Microsoft Azure. Validation of the framework is done by performing iterative experiments with IoT devices set up in rooms on a campus enabled by a private LAN, this campus is based in Helsinki, Finland. For experiments, multivariate time series forecasting is done to predict future air quality in respective rooms using machine learning models deployed in respective edge devices. We conclude these AIoT experiments to validate proposed edge MLOps framework efficiency, robustness, scalability and resource optimization.

Keywords:	Edge Computing, Machine Learning, IoT, 5G Networks, AI, automation, digital transformation
Number of pages:	84
Language:	English
Date of acceptance:	26.05.2020

CONTENTS

FIGURES

TABLES

ACKNOWLEDGEMENT

This thesis is supported by **TietoEVRY** and **5G-Force** project (5G-Force). 5G-Force is part of 5G Test Network Finland (5GTNF 2020). 5G-Force project is funded by Business Finland (Finland 2020). Thank you TietoEvry and 5G-Force project for the support.

FOREWORD

This thesis is a product of a multifaceted one year endeavor from June 2019 to May 2020. This was inspired by the idea of finding a solution to the complex landscape of edge computing, IoT and Machine learning. The thesis intends to examine how a framework that integrates continuous delivery and continuous deployment of machine learning models at the edge can be implemented using state-of-the-art tools and methods. It was done under the teaching supervision of Professor Magnus Westerlund, Arcada University of Applied Sciences and under industry supervision of my colleagues Dr. Ari Rantanen and Dr. Chengyu Liu from TietoEvry.

I want to thank my supervisors, Magnus Westerlund, Dr. Ari Rantanen and Dr. Chengyu liu for being a great help during the development of this thesis. This thesis is done in collaboration with partners TietoEvry and VTT Finland as part of the 5G-Force project and Smart Otaniemi project, both research projects are based in Helsinki, Finland. This project focuses on innovation and building smart solutions for the future. With the inception of the project, we got access to the premises arranged by VTT Finland where IoT devices were set up for our experiments for the project. A key objective of the project is to understand computing trends and curating smart solutions for the future. Special thanks to Dr. Seppo Horsmanheimo, Kimmo Ahola and Sami Huilla from VTT Finland for great collaboration, feedback and support throughout the experiments with the necessary hardware and software support. Lastly, thanks to my family and friends for encouragement and guidance. Especially my mother Blandina Raj, father Ch. Joseph S Raju and friends Anton Akusok, Rita Azevedo, Henrik Meyer and Thomas Forss for supporting me to frame my thoughts and ideas over the last year and a half. This thesis personally enabled me to understand my strengths, weaknesses and formulate a clear vision.

Emmanuel Raj

1 INTRODUCTION

The onset of this millennium resulted in a surge in cloud computing making it a vital part of businesses and IT infrastructures. Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user (Dillon et al. 2010). It offers benefits to organizations such as no need to buy and maintain infrastructure, less in-house expertise required, scaling, robust services and pays as you go features. Especially in the last decade, many global and regional players like Google, Microsoft, Amazon, IBM, Upcloud, Alibaba, and more have emerged. Over the years Cloud computing has revolutionized the way organizations build applications and operate their data centers.

Organizations are now able to centrally store massive amounts of data and optimize computational resources to deliver on their data processing needs which depict the change from localized computing (own servers and data centers) to centralized computing (on the cloud). With the advent of big data, moving devices (self-driving cars, mobiles, etc) and industrial IoT, there is an increasing emphasis on local processing of information in order to enable instantaneous decision-making. So we are witnessing a shift in trend from centralized (cloud computing) to decentralized computing. Edge Computing is the process of performing computing tasks physically close to target devices, rather than in the cloud or on the device itself (Shi et al. 2016). It enables extracting knowledge, insights, and making decisions near the data origin. It is quick, secure, local, and facilitates decentralized processing. Edge computing also enables data confidentiality and privacy preservation on demand as it is becoming essential across multiple industries. The growing amount of data (IoT) and the limitations of cloud computing (for networking, computation, and storage) currently are leading to a decentralized system like Edge Computing.

1.1 Background

There is a paradigm shift in computing approach of resource optimization in terms of energy, efficiency, finance, and human resources to sustain computing resources and infrastructure for organizations to run their services (Bilal et al. 2018) (Raj 2019a). We are at a phase where energy and resource optimization is becoming important, hence we see many investments from the public and private sector going towards building smart solutions and cities which enable smart societies (Bilal et al. 2018). This thesis is done

in collaboration with partners TietoEvry and VTT Finland as part of the Smart Otaniemi project which is a research project based in Helsinki, Finland (VTT 2019). This project focuses on innovation and building smart solutions for the future. With the inception of the project, we got access to premises arranged by VTT Finland where IoT devices were set up for our experiments. This was an ideal setup to begin our experiments to understand computing and infrastructural landscape with a goal to curate smart solutions for the future.

1.2 Developments

Cloud computing was adopted by the industry in the year 2006 with Sun Microsystems introducing hardware and data resource sharing, it was launched as Sun Grid in March 2006 and later named Sun Cloud just prior to being acquired by Oracle. Simultaneously Amazon promoted its cloud computing services as "Elastic Compute Cloud" (Techcrunch 2016). This opened up new opportunities in terms of computation, storage and scaling which attracted attention across industries.

Although cloud computing evolved to go to solutions to many use cases and industries, cloud computing as such was not the solution in all use-cases. With the advent of big data, self-driving cars, and Internet-of-things (IoT), for example, there is an increasing emphasis on local processing of information in order to enable instantaneous decision-making using AI which is also called Artificial Intelligence of Things (AIoT) (Tan & Wang 2010) (Wu et al. 2019). Edge computing unlocks the potential to make real-time decisions or extract knowledge near the data origins (Shi et al. 2016).

1.2.1 Origins

The genesis of decentralized computing can be backtracked to the 1990s when the content delivery network(CDN) was launched by Akamai, since then there have been major developments in cloud computing, edge computing, IoT, and low latency Networks. When Akamai launched its content delivery network. The idea was to introduce nodes at locations geographically closer to the end-user for the delivery of cached content such as images and videos.

In 1997, Akamai's work on "Agile application-aware adaptation for mobility," demonstrated how resource-constrained mobile devices can offload certain tasks to powerful

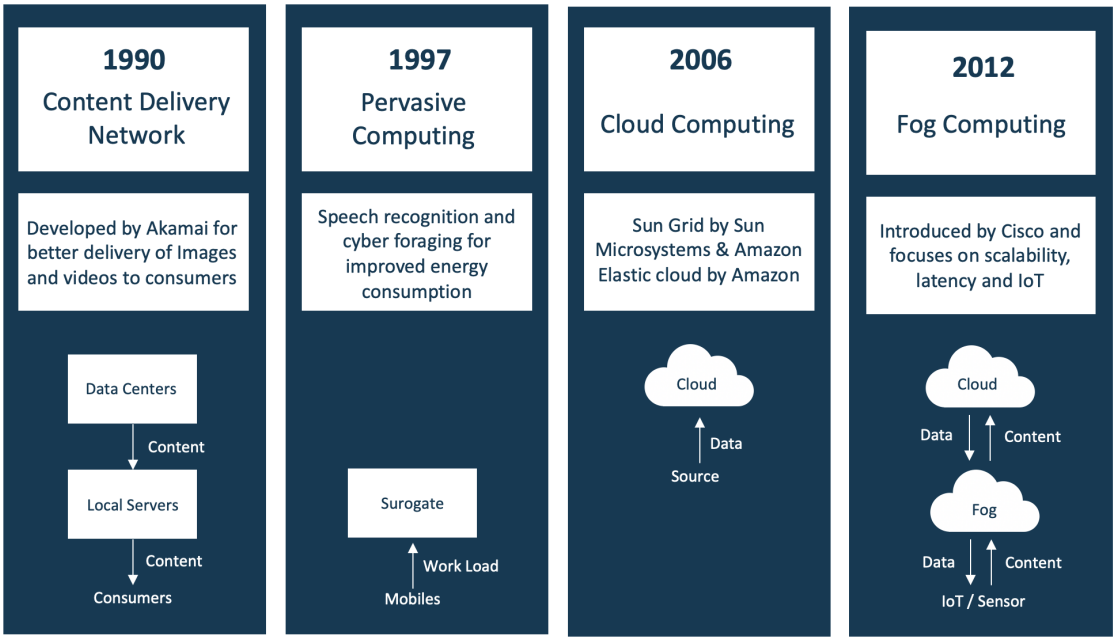

Figure 1. Development milestones of cloud and edge computing

servers especially for different types of applications like web browsers, graphics, video, and speech recognition (Brian D. Noble 1997) . Akamai's work was mainly focused on pervasive computing environment as it facilitates computing and communication capabilities to serve users (Satyanarayanan 2001). For example, today companies like Google, Apple, and Amazon work in a similar way for speech-recognition services. The cloud computing era began in 2006 when Sun Microsystems introduced Sun Grid (later named Sun Cloud) and Amazon introduced its "Elastic Compute Cloud" (Techcrunch 2016). Amazon introduced the pay-as-you-go model (Amazon Web Services) that popularized the use of cloud computing. In telecom we did have cloud computing like solutions before, early 2000s, but not as usage based payments. Amazon Web Services and other subsequent cloud providers have opened up many new opportunities in terms of computation, visualization, and storage capacities. However, Cloud computing comes with some security and data privacy issues and challenges, hence it is important to assess what they provide, limitations, pros, and cons and get an overall understanding of the fast-developing ecosystem (Popović & Hocenski 2010).

1.2.2 Recent Developments

In 2010, Ericsson predicted that 50 billion devices would be connected by 2020, a prediction echoed by Cisco in 2011, as of 2018 there was an estimated 22B devices (Statista 2020). To enable such a scale of IoT devices in low latency edge computing will play an

important role. In year 2012, Edge computing started getting attention with Cisco introducing fog computing for distributed cloud infrastructures (Bonomi et al. 2012). The aim was to promote IoT scalability and robustness, to handle a huge number of IoT devices and big data volumes for real-time low-latency applications. Fog computing shifts data processing to a centralized system on a local area network (LAN) by interacting with industrial gateways and embedded computer systems, whereas edge computing performs data processing on the compute devices directly interfacing to sensors or data origins.

To manage IoT devices, big data and to enable faster decisions, edge computing offers opportunities to take compute close to data origin, it is an ideal choice when it comes to cases with IoT devices, low latency, and real-time operations, For IoT applications to serve at the scale it is important to have the right synergy for cloud and edge. Soon, IoT solutions have to cover a much broader scope of requirements keeping scalability and robustness as the focus. As we see AI integrating into industries, it is vital to synergize edge computing and artificial intelligence to enable real-time decisions near data origins for robust and scalable systems in future.

1.3 Edge Computing

Edge Computing is the process of performing computing tasks physically close to target devices, rather than in the cloud or on the device itself (Shi et al. 2016). It enables extracting knowledge, insights, and making decisions near the data origin. It is quick, secure, local, and facilitates decentralized processing. Edge computing also enables data confidentiality and privacy preservation on demand as it is becoming essential across multiple industries. The growing amount of data from Internet of Things (IoT) and the limitations of cloud computing (for networking, computation, and storage) currently are leading to a decentralized system like Edge Computing.

2 FRAMEWORK

In this thesis, we will implement rapid experiments for IoT and 5G set up on the edge to evaluate offloading machine learning at the edge compared to machine learning centrally at the cloud and its implications. One of the main goals of this thesis is to develop a robust and scalable operational framework for efficient continuous integration and continuous deployment of Machine learning models at the edge for AIoT applications (Wu et al. 2019). Before we delve into the experiment details, setup, and goals, let us look the benefits of using AI and machine learning at the edge in the next section.

2.1 AI and Machine Learning at the edge

The purpose of edge computing is to put computing close to the data source and to offload centralized computing to decentralized. Edge computing makes it possible to apply different machine learning algorithms at the edge, which enable new kinds of experiences and new kinds of opportunities across many industries ranging from Mobile and Connected Home, to Security, Surveillance, and Automotive. It also enables secure and reliable performance for data processing and coordination of multiple devices (Beyer et al. 2018). Figure 2 depicts an overview diagram of how a secure and reliable intelligent edge architecture is constructed.

Figure 2. Intelligent edge and Intelligent cloud powered by 5G networks

Edge computing has several benefits compared to traditional cloud-based computing. For example, researchers built a service to run face recognition applications where the response time is reduced from 900 ms on the cloud to 169 ms by moving the application to edge (Yi et al. 2015). Another example where researchers used edge computing to offload computing tasks for wearable cognitive assistance, resulted in the improvement of response time ranging between 80 to 200 ms which is exponentially better than the central or cloud computing approach (Ha et al. 2014). Also, a by-product of this approach is the energy consumption could be reduced by 30 to 40 % by implementing edge computing (Shi et al. 2016).

The advantages for Machine Learning at the Edge are the following:

1. **Stronger Hardware**: In today's world, many applications rely on very strong or specialized hardware. Modern machine learning algorithms, for example, work best with GPUs or tensor processing units (TPUs). Day by day, edge devices are getting hardware upgrades that enable high computation power to small hardware devices (Girish Agarwal 2019). With upgrades and multiple edge devices, we are able to exponentially increase overall hardware capacity in order to serve and infer robust and scalable machine learning at the edge (near data origins).

2. **Better Latency** (compared to cloud): If applications depend on immediate feedback (e.g. to make "real-time" decisions), sending data to the cloud, calculating and sending the data back to the device may take too long. However, if the path is reduced to the (much closer) edge node and back, many use cases can be realised (Satyanarayanan 2017).

3. **Hyper Personalization**: Devices (IoT) can be in different environments and locations, they might need to perform tasks customised to their respective environments. in such cases edge devices or nodes can enable customisation for each device as in a custom ML model for each device performing realtime inference at close proximity (Girish Agarwal 2019). Also, this way ML models deployed at the edge can optimize and retrain when needed, constantly learn to serve better. This is limited and not possible on scale in the Cloud.

4. **Data throughput**: Devices may produce enormous amounts of data. One single autonomous car for example may produce up to 4000 gigabytes per day (Nelson 2016). If every single car sent all data it generates all the way to central datacenters it would create a huge load on the network. By performing the necessary computations on edge nodes close to the device, most of the paths can be pruned. This is especially important when considering the increasing importance of the internet of things and the rising number of devices connected to the internet (Shi et al. 2016).

5. **Reliability and robustness**: The main functionality of devices should still be available, even if communications to the central cloud are impaired. This can be achieved by relying on local communication with an Edge Node which should (in theory at least) be less prone to problems (Girish Agarwal 2019). If an edge node fails, the devices will be shifted to an alternative edge node.

6. **Privacy**: In many use cases collecting user data is required or useful. However, in cases where aggregated data is sufficient, the users privacy can be preserved by aggregating the data on the edge node instead of the cloud (Westerlund 2018). Edge computing can facilitate data and privacy-preserving machine learning which is also called Federated Learning (Konečnỳ et al. 2016), in which no edge node or centralized compute exchanges data.

7. **Scalability**: In most cases the computing power of devices is limited by their small size. Furthermore, developing a new use case that requires stronger hardware will require all possible users or the network administrator to update the devices, which limits the use cases adoption rate. Edge nodes do not suffer from these problems and can be extended both very easily and continuously (Beck et al. 2014). Using a suitable edge computing framework, adding, replacing or upgrading edge devices is a simple and highly automated process.

8. **Adaptability**: Utility of an edge node instead of a single purpose server has the added benefit of being adaptable to changing circumstances (Girish Agarwal 2019). After enabling a base environment, edge nodes can be easily configured to provide individual subsets of services depending on the environment. Some use cases are only useful in cities while others may be more beneficial in rural areas. Due to the direct connection

to the cloud and higher-level Edge Nodes moving workloads and freeing up computing power for critical use cases is possible and can be done on the fly.

9. **Sustainability and Cost reduction**: Devices are producing enormous amounts of data. One single autonomous car for example may produce up to 4000 gigabytes per day (Nelson 2016). If every single car sent all data it generates all the way to the cloud for machine learning inference it would create a huge load on the network and electricity consumption to serve these requests would be tremendous burden and in turn this would also result in huge costs for Businesses. Instead, outsourcing and taking Machine Learning inference and data pruning at the data origin on the edge will exponentially decrease costs and enable sustainable business (Girish Agarwal 2019).

2.2 Current Problem

In this section, we will observe and reflect on some current problems, needs and trends that are leading us to envision a smarter solution to cater to future needs. Every need or growing problem presents us with opportunities to optimize and improve with the current tools. Let's look into some trends and growing needs over the past years.

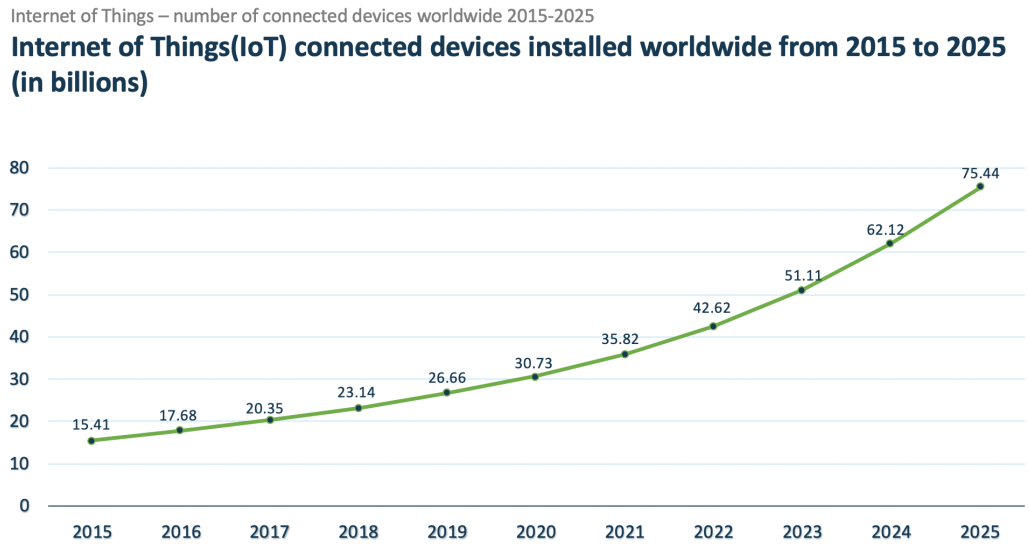

Figure 3. Growth of IoT devices over time.
source: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

In the last decade, we have seen explosive growth in connected devices which is pioneering the world into the era of Internet-of-Things. Figure 3 shows how several IoT connected devices installed worldwide are growing in billions. To handle this growth

cloud computing has its limitations in supporting lightweight IoT devices, especially for the delay in communication context awareness of applications. These applications need to serve a high volume of IoT devices, in realtime for data processing, management of IoT networks and making context-aware decisions in realtime near the vicinity of IoT devices, this becomes necessary for paving the way to address the challenges of cloud computing and the emergence of edge computing (Ren et al. 2017). With this huge potential, some challenges would arise when we do Machine Learning on edge devices at scale to manage and monitor the machine learning models, edge devices, secure devices and their communication, and efficiency of the system and many more questions around it. We can tackle these challenges by taking a systematic approach to enabling continuous deployment and continuous integration at the edge with the right synergy with the cloud. Implementing such practices in your AI projects will yield sustainable and fruitful results. You can produce more productive and robust machine learning models in terms of training, execution time, deploying, and monitoring.

2.3 Solution

To address the problem of the growing number of IoT devices, big data and limitations of centralized or cloud computing, here is a birds-eye view of the process for edge and cloud synergy that will enable robust and scalable Machine Learning at the edge.

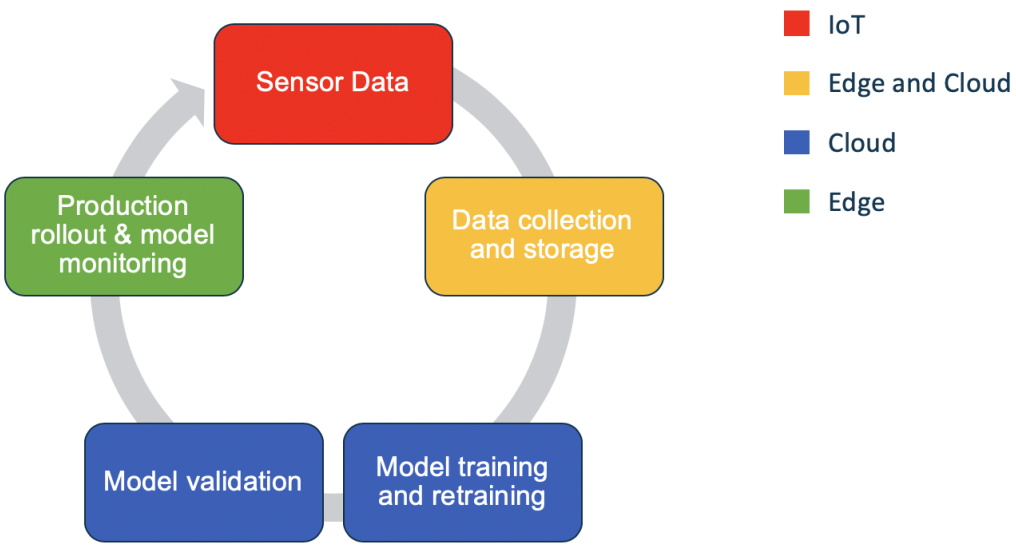

Figure 4. Continuous delivery for Machine Learning on Edge

In this process, sensor data is collected from IoT device(s) by the edge device(s) where the model is inferred (ie: ML model prediction) and the collected data is concatenated with the model prediction and sent in batches for storage at the cloud. Once Data is stored successfully in the cloud, data collected in the edge is deleted.

It is a good practice to continuously monitor the incoming data and retrain your model on newer data based on the deployed machine learning model drift, if you find that the data distribution has deviated significantly from the original training data distribution (Akkiraju et al. 2018). And based on that perform retraining of the machine learning.

Model Training, Validation, and versioning are done on the cloud due to the availability of robust and scalable computation and storage enabled by cloud services. Once a machine learning model is trained or retrained it is rolled out to production deployment at the edge device(s).

We will be looking into executing this process as a solution using best-fit cloud services (from Microsoft Azure) to synergize with edge computers near IoT sensors in a low latency network to perform machine learning inference or decisions in real-time autonomously.

2.3.1 Research Question

The objective of the thesis is to investigate and develop methods that will enable automated edge artificial intelligence. The focus of the research will be to answer the question:

"How can a framework that integrates continuous delivery and continuous deployment of machine learning models at the edge be implemented using state-of-the-art tools and methods."

2.3.2 Studies for addressing the research question

In order to address the research question we will study the following engineering areas. Following are the studies to curate an automated edge AIOps framework for AIoT application:

1. To assess the maturity of cloud services to enable operations on the edge and to identify the limitation.

2. To curate a process for continuous delivery and deployment of Machine learning models at the edge.

3. To explore ways of working in real-time IoT data processing and machine learning.

4. To observe how automated systems operate in real life and production settings.

2.3.3 Significance to the field

This thesis will contribute to the field of machine learning by exploring applied machine learning to modern technologies like edge computing, IoT, and modern networking protocols (MQTT). Contribution to the field as follows,

1. Proposing a flexible architecture that can serve multiple use cases across multiple industries.

2. Assessing the current limitations of cloud services and looking at work around to achieve the most efficient ways of working for automated edge AI.

3. Validate benefits of applied machine learning at the edge such as better latency, reliability, robustness, hyper personalization, sustainability and cost reduction.

2.4 Components and processes for edge AI ecosystem

This sub-section introduces components and processes that are central to solving the research question.

- Edge Computing: Edge Computing allows data to be analyzed near or at the location of data origin before being sent to the cloud or data center. At the edge, knowledge can be extracted and decisions can be made using AI.

- Machine Learning: Machine learning is a method of data analysis that automates analytical model building, Machine learning algorithms can learn from data, identify patterns and make decisions without much human involvement.

- Continuous Delivery and Deployment: Continuous delivery is focused on keeping software releasable all the time, continuous deployment extends it to continuously and automatically deploy new changes into production. A continuous deployment is a push-based approach, by which code changes are automatically deployed to a production environment through a pipeline as soon as they are ready, without human intervention. Continuous delivery is a pull-based approach in which a person (e.g., a manager) is required to decide which and when production-ready code changes should be released to production (Shahin et al. 2019).

- Fleet Analytics: It is aggregated analytics for each edge device used in the experiment which depicts device performance over a period of time with telemetry data like accelerometer, gyroscope, humidity, magnetometer, pressure and temperature. This information in turn is useful to monitor edge devices health and longevity.

- Machine Learning Lifecycle Management: Machine Learning lifecycle management is an efficient way of working for building, deploying, and managing machine learning models critical for ensuring the integrity of business processes (Raj 2019b).

- AIoT: Artificial Intelligence of Things (AIoT) is the combination of artificial intelligence (AI) technologies with the Internet of Things (IoT) infrastructure to achieve more efficient IoT operations, improve human-machine interactions, enhance data management, analytics and decision making (Wu et al. 2019) (Rouse 2020).

- MQTT (Networking protocol): MQTT is based on clients and brokers, were the client's requests of receiving or sending data between each other (e.g. Edge devices) and broker (like a server). The broker is responsible for handling the client's requests for receiving or sending data between each other.

  1. The MQTT server is called a broker and the clients are simply the connected devices, in our case it is IoT sensors.

  2. When a device (a client) wants to send data to the broker, we call this operation a "publish". When a device (a client) wants to receive data from the broker, we call this operation a "subscribe".

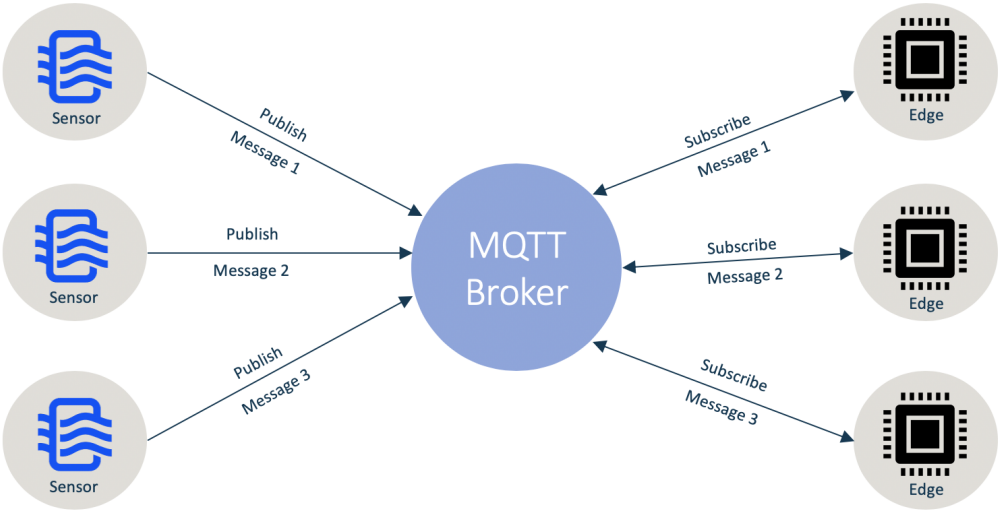

Figure 5. Schematic data flow and communication from sensor machine to edge device

MQTT is designed as a robust, session-oriented protocol especially suitable for the world of IoT, where the clientID plays the central role for session management. The MQTT specification requires the clientID to be provided within the first data frame of the protocol during session establishment. The semantics of the clientID is to provide the unique way a session can be (re)established between a client and the broker, without any further information. So the clientID is required to be unique per broker over time, hence, no collision of clientIDs should ever happen. As the clients are not aware of each other, but usually provide their own clientID, it must be drawn from large set of possible clientIDs so the probability of a collision of clientIDs is negligible.

2.5 Limitations

Edge computing has many advantages (as discussed in section 2.1), but also has certain limitations. Some limitations of edge computing are shown in figure 6 .

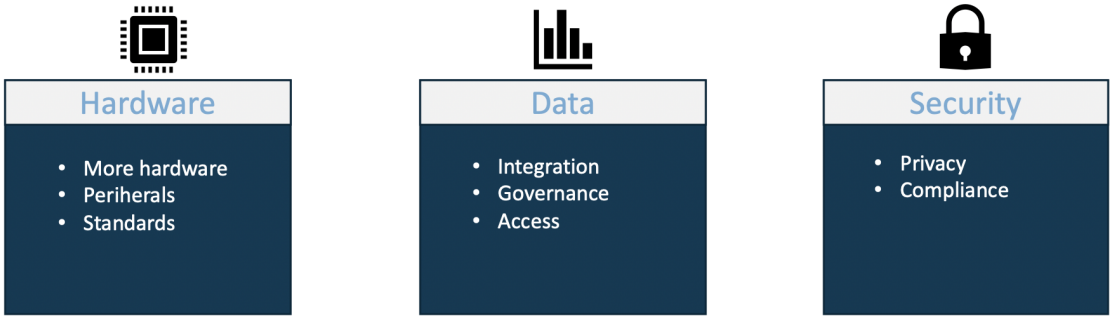

Figure 6. Limitations of edge computing.

1. More Hardware: Edge computing requires setting up more local hardware. Eg: IoT cameras on the street or self driving cars require a built-in compute hardware to process, infer and send video data over the internet to the cloud as well as a more sophisticated computing process for more advance process applications, such as objects-detection, motion-detection or facial-recognition algorithm. So for adoption of edge computing on scale a massive add on to existing infrastructure is needed in terms of more computing power.

2. Data limitations: Edge devices store, processes and analyzes only a subset of data, discarding raw information and sending only needed information to the cloud. Organizations must consider what level of loss of data is acceptable and have a solid data strategy in place for edge computing. Contrary to this data pruning when done right can be a major benefit. Edge devices also have limited access to full data in cloud or on-sight and it's data governance is limited to edge device only.

3. Security: Edge computing can increase the probability of attacks. With IoT sensors, networking, and built-in computing the chances of attacks by malicious hackers to infiltrate the devices and access sensitive data have increased. One potential risk is data and device manipulation attacks. If hacked, it is possible to manipulate the device about the data it has collected, leading to bad decisions. Security is an important area in edge computing, in most of the use cases privacy and compliance are highly important so edge-cloud setup has to comply with data privacy and laws for location (Beck et al. 2014).

2.6 Aim of the project

The aim of the project is to evaluate, validate or develop a robust and scalable framework for edge computing that will enable automated machine learning at the edge for AIOT applications, The application will be fairly industry and use case agnostic. This framework would facilitate:

- Continuous integration and continuous deployment of Machine Learning models at the edge.
- Fleet analytics to monitor edge devices in real-time.
- Machine learning model lifecycle management.

3 RESEARCH THEORY AND METHODOLOGY

Studies in software engineering are often of an interdisciplinary nature and this thesis is not an exception. The research field of software engineering is often defined as an intersection of information technology, business, and data processing. In this study, the business dimension is focused on private and public companies determined to optimize resources, increase efficiency and upgrade their existing services to be smarter and enabled by the power of artificial intelligence through real time data using Internet-of-Things. For the social context of the research these businesses have goals to optimize resources in terms of energy, time, money and human resources. Optimizing these resources will drive efficiency, growth and adoption of modern technologies like edge computing, artificial intelligence of things (AIoT) and low latency networks.

In order to address our research question we will follow a design science method proposed by Wieringa (2014). Design science is the design and investigation of artifacts in a context. The artifacts we study are designed to interact with a problem context in order to improve something in that context.

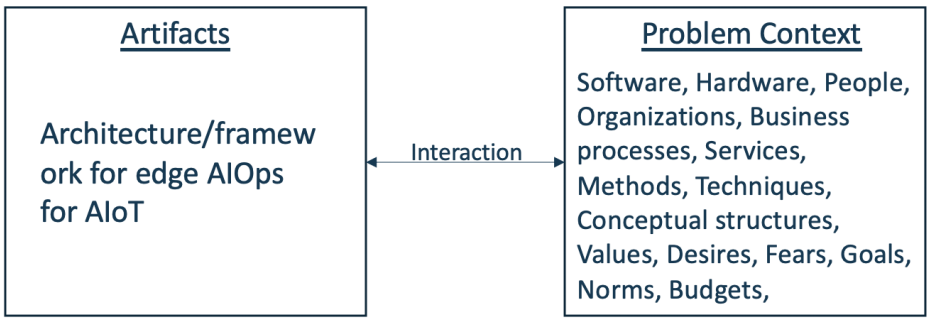

Figure 7. The subject of design science: an artifact interacting with a context

In our case the artifact in context will be the edge MLOps framework for AIoT applications that we will design iteratively while interacting with the problem context which boils down to our project or experiments we perform in order to evaluate and validate the scalability and robustness of the artifact in the context. The project and experiments we will work on replicates software, hardware, people, organizations, business processes, services and values of the real world setup with similar social context as discussed above (VTT 2019).

3.1 Towards a Research Methodology using Design Science

The research method used follows the outline of design science. In a structured and iterative approach, we implement two cycles (Design cycle and Empirical cycle) for qualitative and quantitative analysis and conclusions for our design solution.

3.1.1 Design Cycle

Design problems call for a change in the real world and require an analysis of actual or hypothetical stakeholder goals. A solution is a design, and there are usually many different solutions. There may even be as many solutions as there are designers. These are evaluated by their utility with respect to the stakeholder goals, and there is not one single best solution. For example, what is an accurate algorithm for image classification? there isn't a one go to solution for image classification but instead there are different algorithms, designs and ways of classification. It is about finding the right algorithm or way of working for the current problem. We can have an ideal design to address a problem context but not a one and the only design. Ideal design to solve a problem in a generalized approach is achieved by iterative experiments on problem to find ideal solution by assessing the utility value for the problem context.

To design an ideal workflow or ecosystem for our problem context (AIoT on edge computing) its application and utility will be industry and use case agnostic. This will be achieved by exploring and building through services available on a popular cloud service Microsoft Azure. This ecosystem would facilitate,

- CI/CD of Machine Learning models at the edge
- Fleet Analytics
- Machine learning model lifecycle management

To iterate in design cycle we will work on the following problem context (experiment): Predict room air quality for anomaly detection using Machine Learning on the edge. Experiment setup has 3 rooms with an edge computer in each (3 edge computers in total) upon receiving data from sensors, edge devices should make machine learning model in-

ference to predict air quality in next 15 minutes, this process or flow has to be automated using state-of-the-art tools and methods.

3.1.2 Empirical Cycle

Empirical cycle which is a rational way to answer scientific knowledge questions. It is structured approach for qualitative and quantitative analysis and conclusions for our design solution.

Below is a set of question in form of a checklist to decide the success of our design solution. Goal is to get the optimal results in an iterative process.

1. Research problem analysis: To investigate an improvement of problem in the field.
    - To explore ways of working in real-time with IoT data processing and machine learning.
    - To explore methods for applied machine learning for real-time multivariate time series forecasting.
    - To observe how automated systems operate in real life and production settings.
2. Research design and inference design: To survey possible methods.
    - To assess the maturity of cloud services to enable operations on the edge and to identify the limitation.
    - To curate a process for Continuous delivery and deployment of Machine learning models at the edge
    - Machine Learning lifecycle management design for edge AI.
3. Validation of research and inference design:
    - Robustness: Stability of CI/CD pipeline (Number of successful triggers) for cloud to edge, Stability of CI for IoT devices to edge devices, Hardware compatibility, Model performance (number of models changed, model drifts), models retrained (Success and failed), fleet analytics and data storage.

- Scalabilty: Number of Devices, Multiple Cloud, Network scaling.
- Application: Industries and use cases agnosticity.
- Resources Optimization: Optimization of Energy, Time, Human interaction and Cost compared to cloud computing (current setup).

3.2 Applied Machine Learning Methods

We look at the problem of predicting a single variable (future air quality - 15 minutes in the future) using multiple independent variables. In this section we explore the Machine Learning methods applied on the data to develop models that will be deployed on the edge devices to make real-time prediction. Let's look at them one by one.

1. Multiple Linear Regression (MLR)

Multiple linear regression, models the relationship between two or more explanatory variables in correlation to a response variable. In order to understand the correlation better, we segregate all the variables into two categories namely, independent variables and dependent variable

INDEPENDENT VARIABLES (X) : Variables or factors which are used to correlate to response or prediction or dependent variable.

DEPENDENT VARIABLES (Y): The outcome variable is called the dependent variable.

Every value of the independent variable x is associated with a value of the dependent variable y (Preacher et al. 2006). The correlation between the independent and dependent variables is calibrated by a regression line in an n-dimensional space for all independent variables. let's say there are n independent variables x1, x2, ... , xn. To predict y or the independent variable, correlation is defined to be

$$y = \beta_0 + \beta_1 X_i + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_n X_n \tag{1}$$

This line describes how the mean response y changes with the independent variables. The observed values for y vary about their means y and are assumed to have the same standard deviation. The coefficients

$$\beta_0, \beta_1, \beta_2, \beta_3, ...\beta_n \tag{2}$$

are fitted or calibrated to estimate the parameters 0, 1, ..., n of the regression line in order to predict the independent variable y.

2. Extreme Learning Machines

The extreme learning machine has demonstrated excellent performance in a variety of machine learning tasks including situations with missing values. Extreme learning machine is a single layer feedforward neural network with randomly generated neurons for regression, classification, clustering, sparse approximation, compression, and feature learning (Akusok et al. 2015). In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model.

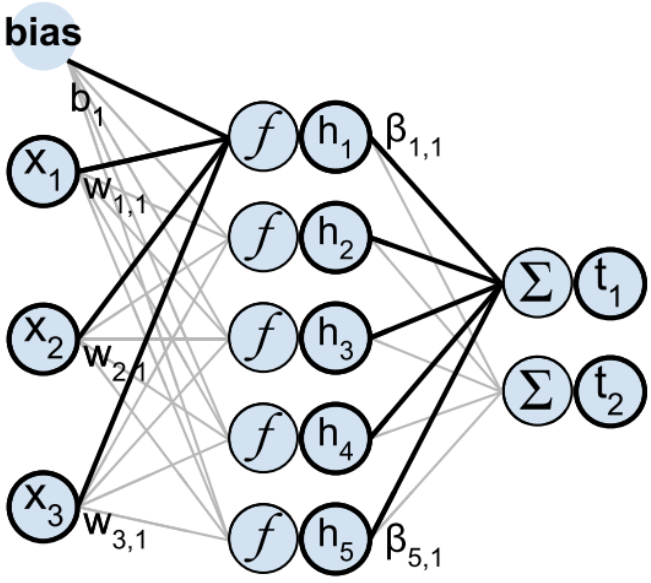

Figure 8. Computing the output of an SLFN (ELM) model
Printed with permission from author Anton Akusok (Akusok et al. 2015).

A hidden layer randomly generates loosely correlated hidden layer features, it allows for a solution with a small normalization and a good generalized performance.

A mathematical description of an ELM is as following. Consider a set of N distinct training samples (x_i, t_i), i ∈ [[1, N]] with $x_i \in R^d$ and $t_i \in R^c$. Then a single hidden layer feed forward network with L hidden neurons has the following output equation:

$$\sum_{j=1}^{L} \beta_j \phi(w_j x_i + b_j), i \in [1, N] \tag{3}$$

with ϕ being the activation function, Sigmoid function is a common choice, but other activation functions are also possible like linear, tan-sigmoid, sin etc (Huang et al. 2011) (Huang 2014) (Huang 2015). w_i the input weights, b_i the biases and β_i the output weights. The relation between inputs xi of the network, target outputs t_i and estimated outputs y_i is:

$$y_j = \sum_{j=1}^{L} \beta_j \phi(w_j x_i + b_j) = t_i + \in_i, i \in [1, N], \tag{4}$$

where $\in$ is noise. Here the noise includes both random noises and dependency on variables not presented in the inputs X (Akusok et al. 2015).

3. Random Forest Regressor Random forest is a type of ensemble learning with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging (Segal 2004). Ensemble Learning is when you take multiple algorithms, combine their output to archive a better combined result than the original (Zhang & Ma 2012).

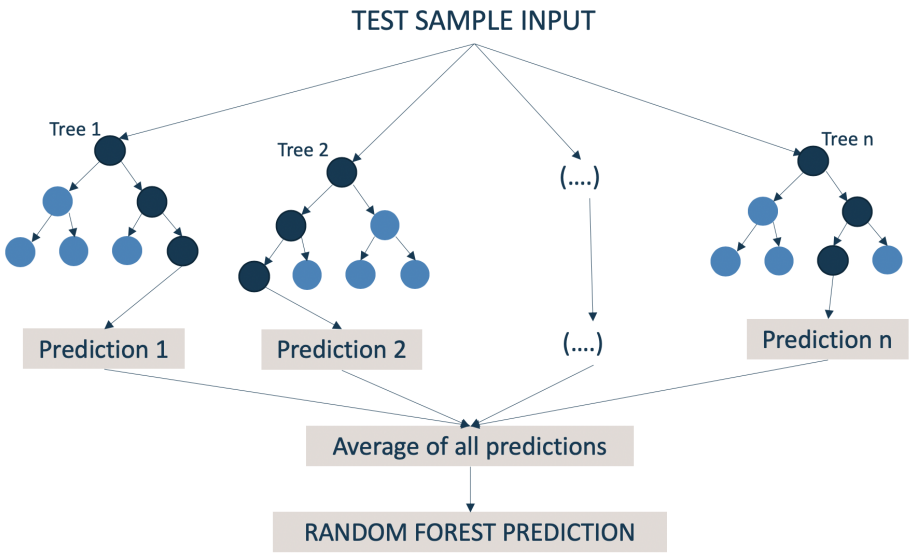

Figure 9. Random forest structure

Random forest is a bagging technique. There is no interaction between these trees while building the trees. The trees in random forests are run in parallel (Liaw et al. 2002).

These are the steps to build a random forest regressor

- Step 1: Pick k random points from the training dataset.
- Step 2: Build the decision trees associated with these K data points.
- Step 3: Choose N number of trees you want to build and repeat steps 1 and 2.
- Step 4: For the new data point or test input, make each one of your N trees predict the value of y for the data point in question, and assign the new data point the average across all of the predicted y values.

Let's say y is the dependent variable to predict and x1, x2, x3 .... xn are independent variables, then we predict y by making each of N number of trees predict the value of y and then average all predictions to derive final prediction for y.

$$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{y_1 + y_2 + \cdots + y_n}{n} \tag{5}$$

4. Support Vector Regressor

The regression problem is a generalization of classification problem, in which the model returns a continuous-valued output, as opposed to an output from a finite set (Ratkowsky & Giles 1990). In other words, a regression model estimates a continuous-valued multivariate function. SVMs solve binary classification problems by formulating them as convex optimization problems. The optimization problem entails finding the maximum margin separating the hyperplane, while correctly classifying as many training points as possible. SVMs represent this optimal hyperplane with support vectors (Drucker et al. 1997). The sparse solution and good generalization of the SVM lend themselves to the adaptation to regression problems. SVM generalization to SVR is accomplished by introducing an ε-insensitive region around the function, called the ε-tube. This tube reformulates the optimization problem to find the tube that best approximates the continuous-valued function while balancing model complexity and prediction error. More specifically, SVR is formulated as an optimization problem by first defining a convex ε-insensitive loss function to be minimized and finding the flattest tube that contains most of the training instances. Hence, a multiobjective function is constructed from the loss function and the geometrical properties of the tube.

Then, the convex optimization, which has a unique solution, is solved, using appropriate numerical optimization algorithms. The hyperplane is represented in terms of support vectors, which are training samples that lie outside the boundary of the tube. As in SVM, the support vectors in SVR are the most influential instances that affect the shape of the tube, and the training and test data are assumed to be independent and identically distributed, drawn from the same fixed but unknown probability distribution function in a supervised-learning context (Gunn et al. 1998).

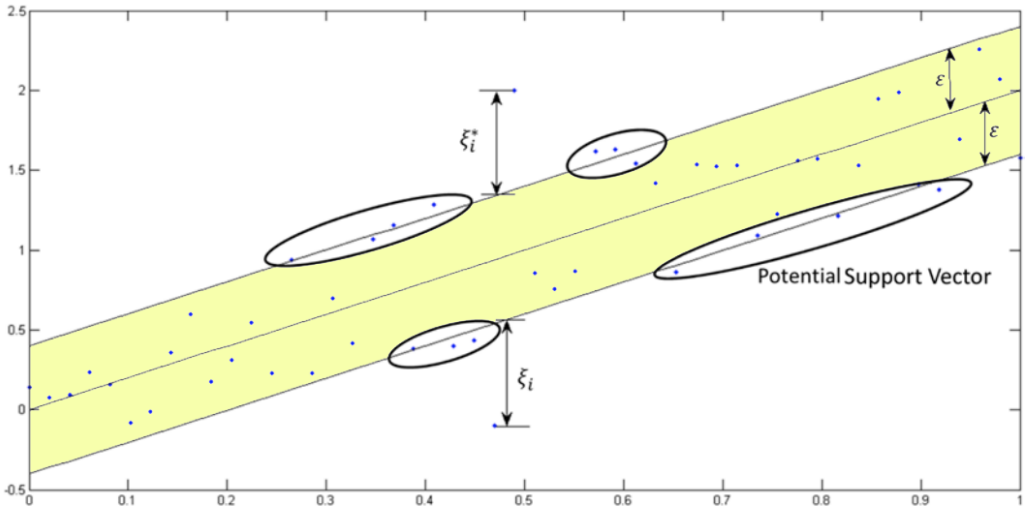

Figure 10. One-dimensional linear SVR

This is a one-dimensional view of an optimized ε-insensitive tube for data points with potential support vectors. ε_i and ε_i^* are the slack variables with ε_i being variables representing training data points above the ε insensitive tube and ε_i^* is for data points below ε insensitive tube.

$$\frac{1}{2}||\omega||^2 + C\sum_{i-1}^{m}(\varepsilon_i - \varepsilon_i^*) \rightarrow min \tag{6}$$

SVR formulates this function approximation problem as an optimization problem that attempts to find the narrowest tube centered around the surface while minimizing the prediction error, that is, the distance between the predicted and the desired outputs. The former condition produces the objective function, where ||ω|| is the magnitude of the normal vector to the surface that is being approximated.

4 EXPERIMENTS

The study is performed through a collaboration project - Smart Otaniemi together with TietoEvry (industry partner) and VTT Finland (research partner). To perform experiments for the thesis, there was a need for infrastructure (IoT and 5G) and a platform to experiment with AI on the edge and IoT devices in real-time. Partners provisioned needed infrastructure and platform to perform experiments in VTT's 5G campus in Helsinki, Finland.

4.1 Setup

At the 5G campus, there are 26 rooms with 26 IoT devices monitoring room air quality and conditions 24x7. Each sensor collects and sends data to the central network/database or station on an interval of 5 minutes.

In the execution of the experiment for the thesis, we narrowed it down to three rooms as shown below in figure 11. The reason for selecting these rooms is described in the Data Analysis section.

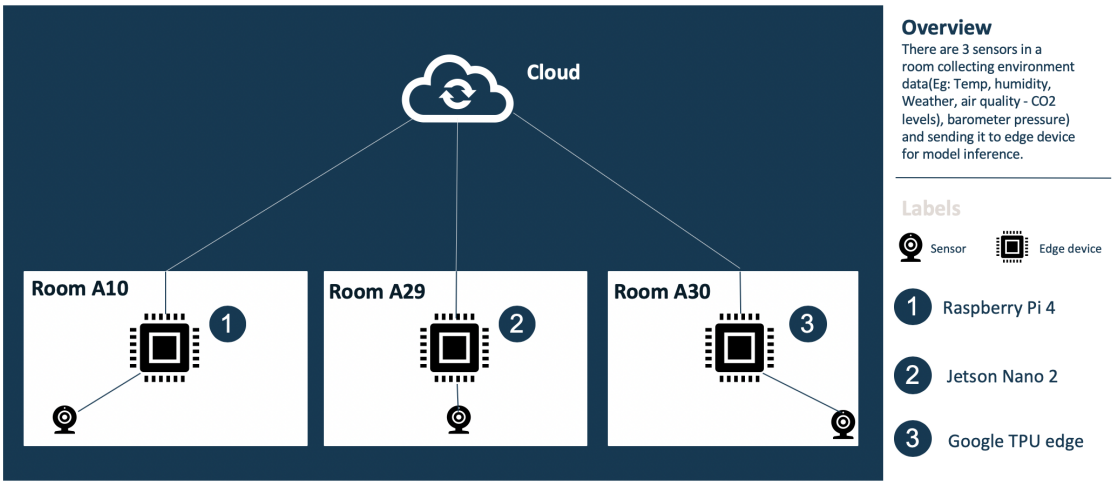

Figure 11. On premises experiment setup

4.1.1 Hardware Tools

In order to implement this setup, following hardware is used for edge computers:

- NVIDIA Jetson Nano2: https://www.nvidia.com/jetson-nano/
- Google TPU edge: https://cloud.google.com/edge-tpu/
- Raspberry Pi 4: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

Raspberry Pi 4 was setup in room A10, Nvidia Jetson Nano 2 was setup in room A29 and Google TPU edge was setup in room A30. Detailed steps of setup and installation for each device listed in Appendix A.

4.1.2 Software Tools

For software development, I have chosen these common data scientist's tools for the tech stack to do data analysis, model training and deployment, and monitoring - python, linux, and docker. The programming language used to conduct experiments for this thesis is Python (version 3.6.7). Numerous libraries are used for different purposes to assist experiments. Find the list of the major libraries used in Appendix B.

For Deployments - Docker containers are used to deploy applications in runtime, a docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. For cloud - Microsoft Azure is used. Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. All of these computer systems, middleware and services need to be arranged and coordinated in such a way that they automated multiple tasks and systems, this process is called as orchestration. Orchestration takes advantage of multiple tasks that are automated to automatically execute a larger workflow or process. The goal of orchestration is to streamline frequent and repeatable processes to ensure optimization and efficient deployment of software. To achieve efficient edge and cloud synergy, services from Azure cloud are used to orchestrate edge to cloud operations for continuous delivery and deployment.

4.2 Machine Learning Operations for AIoT Application

This section describes the systematic approach to Machine Learning operations (MLOps) for data collection, exploratory data analysis, feature extraction and machine learning models training done before our experiments for AIoT applications in real-time. To train we need computation and storage resources and on top of that a platform to train the machine learning models. For this purpose we use Azure Machine Learning service as a platform where we can provision compute, storage and needed infrastructure on request. It is a framework providing an end-to-end solution for machine learning model development as follows:

- Resource provisioning
- Data versioning
- Model training
- Model storing and versioning
- Model packetizing
- Model deploying
- Monitoring

With these features we will be able to train, manage, deploy and audit models (model traceability for data and source code used to train). Models are trained separately to be deployed in respective rooms and edge devices.

4.2.1 Dataset Analysis

This section describes the data that will be used in the experiments to train the machine learning models to be deployed in the edge devices to carry out the experiment and evaluate. The data has been collected for 3 months, starting from 15th October 2019 to 15th January 2020 from 26 different IoT devices.

4.2.1.1 Data descriptors -

Here are the data descriptors for data collected from IoT devices.

1. *timestamp - Time of data (datetime)*
2. *name - Name of sensor (str)*
3. *room - The room where sensor is placed or data origin (str)*
4. *room type - Type of room (str)*
5. *floor - Floor where data was generated (str)*
6. *air quality - Air quality index altered (float)*
7. *air quality static - Air quality index unaltered (float)*
8. *ambient light - Light present in the room (float)*
9. *humidity - Humidity in the room (float)*
10. *iaq accuracy - Indoor Air Quality accuracy altered (float)*
11. *iaq accuracy static - Indoor air quality accuracy un altered (float)*
12. *pressure - Pressure in the room (float)*
13. *temperature - Temperature in the room (float)*

- **air quality and air quality static**: Air quality and air quality Static are air quality indexes in the room ranging from 0 to 250. Air quality static is raw sensor reading and air quality is augmented data. Air quality is hazardous for humans in the range 150-250 (Coway 2016). Air quality is augmented data of air quality static. air quality static is the raw reading for IoT device sensors.

- **ambient light**: Ambient light is the measurement of ambient light intensity that matches the human eye's response to light under a variety of lighting conditions.

- **humidity**: Humidity measures and reports both moisture and air temperature. The ratio of moisture in the air to the highest amount of moisture at a particular air temperature is called relative humidity. Units measured by the sensor are grams per cubic meter. Humidity ranges from 0 to 50, anything above 40 grams per cubic meter can be uneasy for human activity in the room.

- **iaq accuracy and iaq accuracy static**: One of the factors to calibrate indoor air quality (IAQ) is iaq accuracy. IAQ Accuracy=1 means the background history of the sensor is uncertain. This typically means the gas sensor data was too stable to clearly define its references, IAQ Accuracy=2 means sensor found a new calibration data and is currently calibrating, IAQ Accuracy=3 means data calibrated successfully. IAQ accuracy is augmented data and iaq accuracy static is the raw reading for

IoT device sensors.

- **pressure**: Pressure in the room is measured in kpa ranging from 0 to 1040.
- **temperature**: Temperatures in the room have been measure between 0-26 °c.

Here is a snapshot of the raw data collected from IoT devices.

timestamp	name	room	room_type	floor	air_quality	air_quality_static	ambient_light	humidity	iaq_accuracy	iaq_accuracy_static	pressure	temperature
2019-10-16 11:45:12	T009	A09	office_room	A	31.0	27.0	10.0	34.32	1.0	1.0	1010.0	21.57
2019-10-16 11:45:12	T010	A10	office_room	A	64.0	42.0	82.0	33.93	1.0	1.0	1010.0	22.95
2019-10-16 11:45:28	T017	A17	office_room	A	25.0	25.0	123.0	34.71	1.0	1.0	1010.0	22.52
2019-10-16 11:45:40	T019	A19	office_room	A	51.0	40.0	1.0	34.71	3.0	3.0	1009.0	22.00
2019-10-16 11:45:45	T020	A20	office_room	A	59.0	39.0	6.0	33.15	1.0	1.0	1010.0	22.04

Figure 12. Data snapshot of 3 months of data collected from IoT sensors

Here is an overview of the data collected from IoT devices,

- Timeline - 3 months (15-10-2019 to 15-01-2019)
- Total 537873 number of rows or events were recorded.
- Size of the data: 45.9 MB.

Each IoT device generated an event or recorded data at a time interval of 5 mins which equals to 12 events in an hours.

4.2.1.2 Stationarity analysis After assessing time series of air quality for each room's data, figure 13 shows non-stationary pattern since mean, variance and covariance are observed to be changing over time. Non-stationary behaviors can be trends, cycles, random walks or combinations of the three as observed in figure 13.

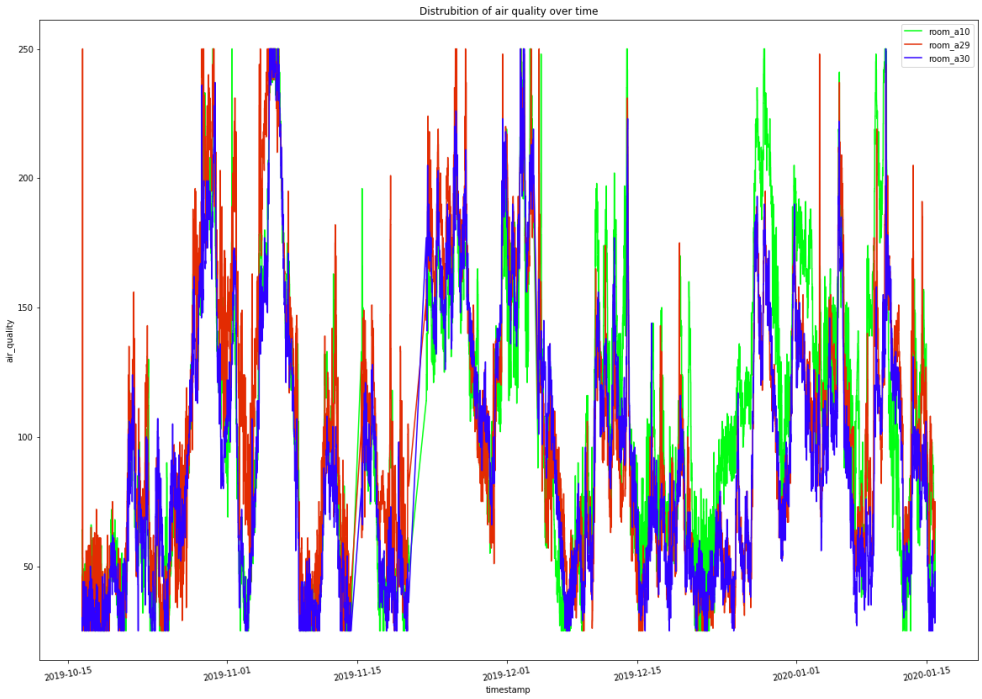

Figure 13. Data non-stationarity over time observed for selected rooms.

There are 26 rooms, each room has an IoT device to monitor room conditions. Over the period in which data was collected, each room has around 46000 events recorded as shown in figure 14.

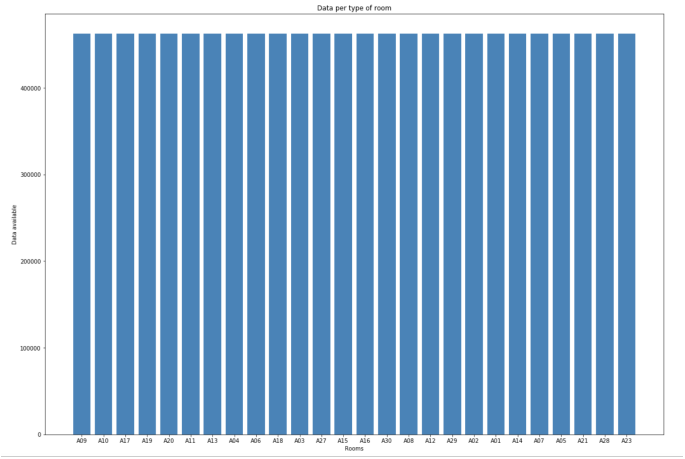

Figure 14. Frequency of data in each room.

There are three types of rooms in the premises, most of them are being office rooms. Rest are meeting rooms and corridor rooms. Here is the collective data frequency for each type of room described in figure 15.

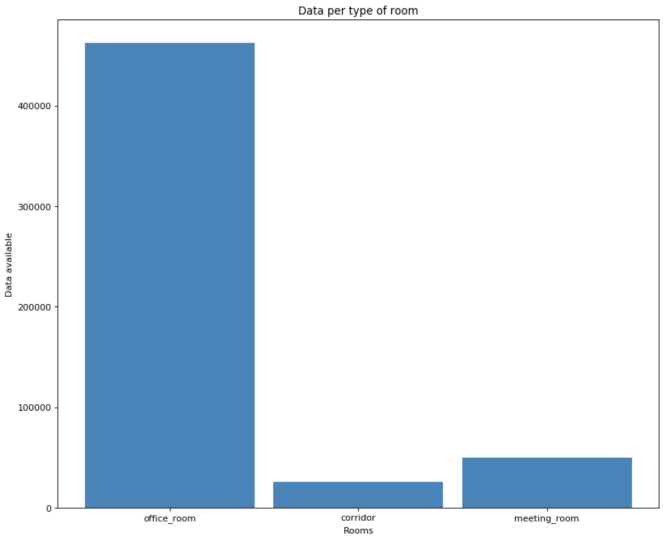

Figure 15. Frequency of data in each room.

After the data analysis, we narrowed down the experiment to only 3 different rooms. Reason being we wanted to experiment on meeting rooms, since we had only 2 meeting rooms available, they were chosen. And one office room was chosen with highest frequency of unhealthy air quality. These are the selected rooms: room_a10, room_a29, room_a30.

Here is the normal distribution of air quality in selected rooms as shown in figure 16.

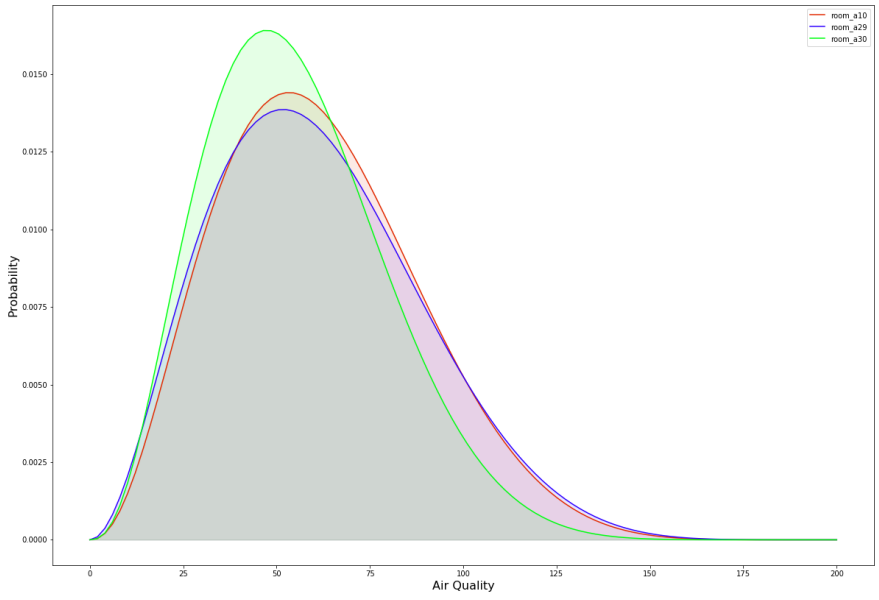

Figure 16. Normal distribution of air quality in rooms

Table 1. Descriptive statistics for air quality in selected rooms.

Selected Rooms			
Room name	Room type	Unhealthy air quality frequency	Avg. air quality
Room A10	Office room	2033	61.92
Room A29	Meeting Room	2205	61.40
Room A30	Meeting Room	1085	55.45

Descriptive statistics for rooms are listed in table 1 with unhealthy air quality frequency and Avg. air quality for each room.

4.2.1.3 Empirical data analysis for selected rooms

In order to assess room conditions and anomalous behavior. Let us look at data in detail for each room in this order room a10, room a29 and room a30.

1. Room a10 - Office room

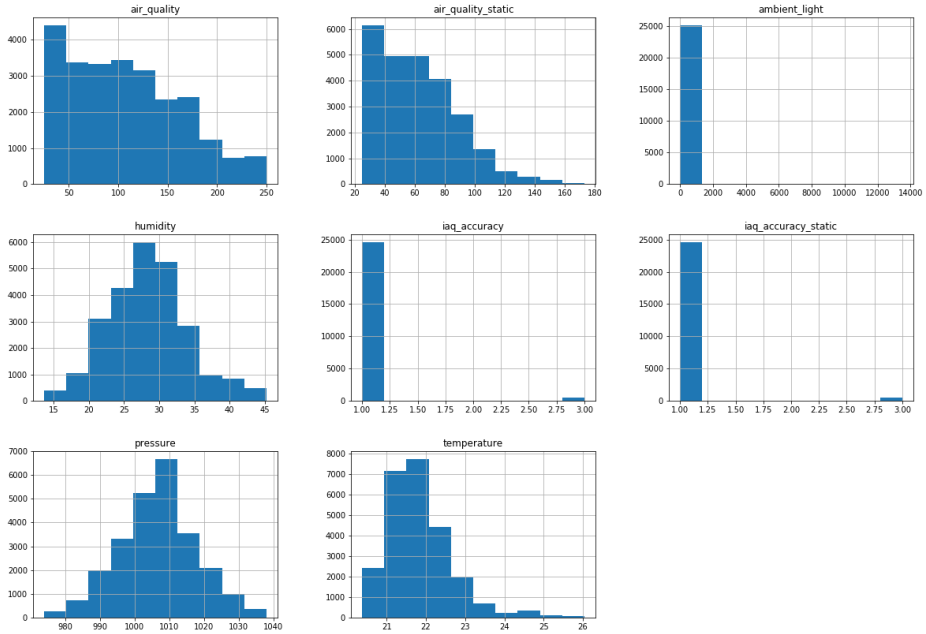

Figure 17. Emprical analysis for room a10.

Histograms for all columns in the data for room a10 are generated (in figure 17) to get a holistic view of data and observe overall conditions and anomalies.

For both, the majority of data points for air quality and air quality static range in good air quality (ie: 0-100) which is a good sign as it shows quality of the air in the room is good majority of the time. Quality of the air is also observed to have some anomalies or worse (ie: 150-250) on some occasions, this range is hazardous for humans in the room (Coway 2016). Ambient light is in two extremes either 1 or 25-30. Average humidity in the room is observed to be in the range 25-35 grams per cubic meter which is healthy for humans, anything above 40 grams per cubic meter can be uneasy for human activity in the room. IAQ accuracy is mostly 1 with some cases of 3 and very few samples of 2. In most cases, the pressure is between the range 990 to 1020. The majority of data points for temperature are ranging in 21-24 °C which is optimal room temperature. Some anomalies have been observed with low temperatures as below 10 °C and above 25 °C. In figure 18, a time-series sensor data for room a10 data progression over time can be observed in these graphs.

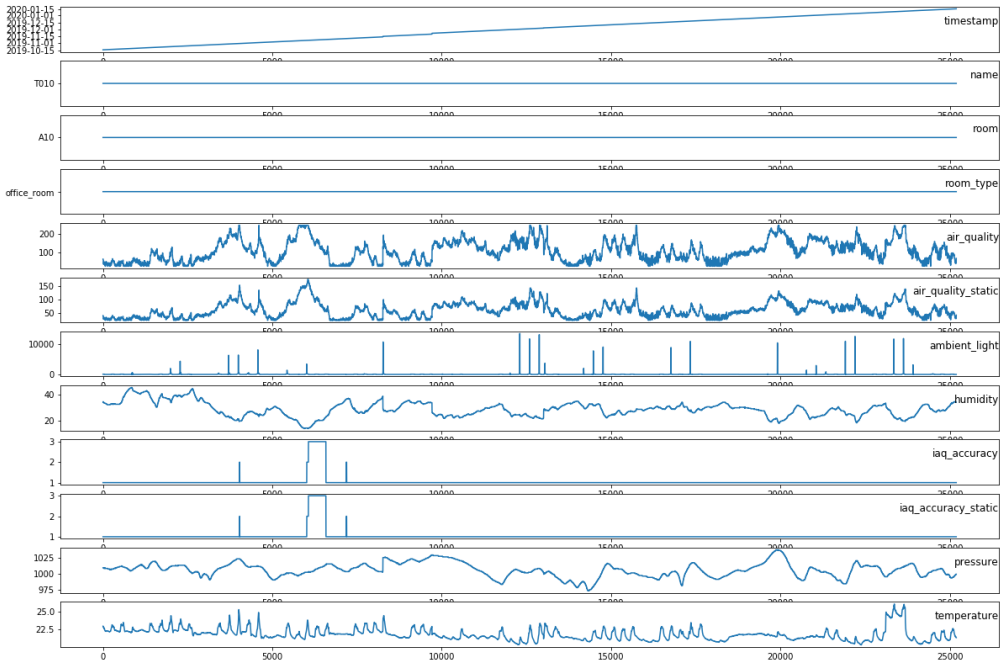

Figure 18. Timeseries data progression for room a10

Air quality and air quality static have identical progressions of data over time and the data is non-stationary. Likewise, humidity, pressure and temperature are observed to be non-stationary and independent of each other.

Some anomalies and peaks are noticeable for ambient light, IAQ accuracy and IAQ accuracy static. For our experiment, we predict air quality static using machine learning at the edge devices. Figure 13 shows how air quality static has progressed over time. Some anomalies and peaks have been noticed for the timeline of 25-10-2019 to 01-11-2019, 01-12-209 to 7-12-2019 and 28-12-2019 to 10-01-2020. Upon cross-checking with the premises authorities they have validated these peaks to be the busiest time during this time of the year where they have a high amount of human activity, i.e. meetings in our case for room a10. These peaks in data are useful for our machine learning models to learn and predict. The average air quality in room_a10 is 61.92.

2. Room a29 - Meeting room

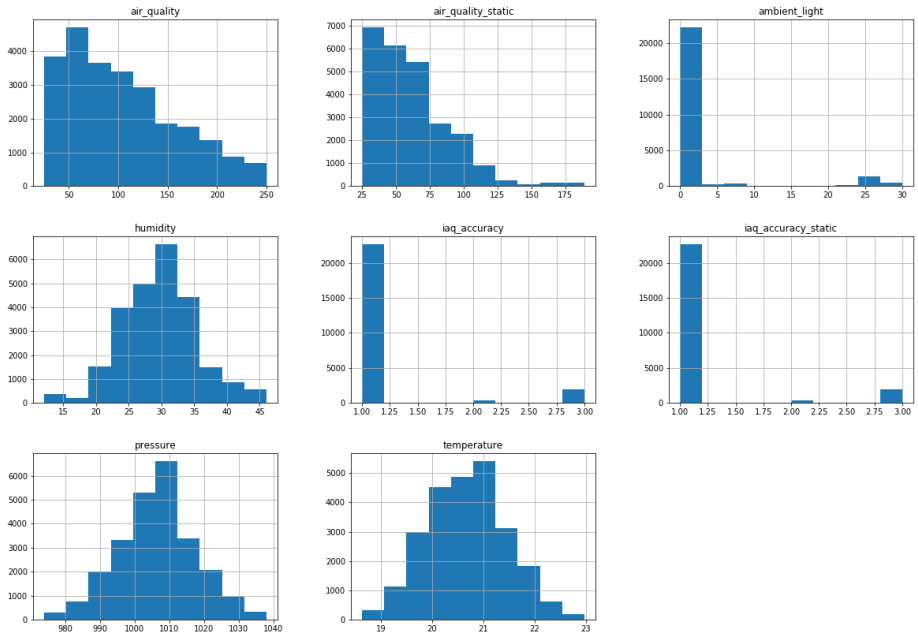

Figure 19. Emprical analysis for room a29

To get a holistic view of data and observe overall conditions and anomalies, histograms are generated for all columns in the data for meeting room a29 (in figure 19). For both air quality and air quality static, the majority of data points range in good air quality (ie: 0-100) which is a good sign as it shows air quality in the room is a good majority of the time. In few instances both are observed to have some anomalies or worse (ie: 150-250), this range is hazardous for humans in the room. Ambient light is in two extremes either 1-9 or 25-30, the majority being in 1-2.

In many instances humidity is observed to be in the range of 25-35 grams per cubic meter which is healthy for humans. IAQ accuracy is mostly 1 with some cases of 3 similar to room_a10. Data observed in the histograms shows IAQ accuracy and IAQ accuracy static are identical. In most events pressure is between the range 1000 to 1020 kpa. In most of the instances, the temperature is ranging between 21-24 °C. Some anomalies have been observed now and then with low temperatures as below 10 °C and above 25 °C. In figure 20, a time-series sensor data for room_a29 data progression over time can be observed in these graphs.

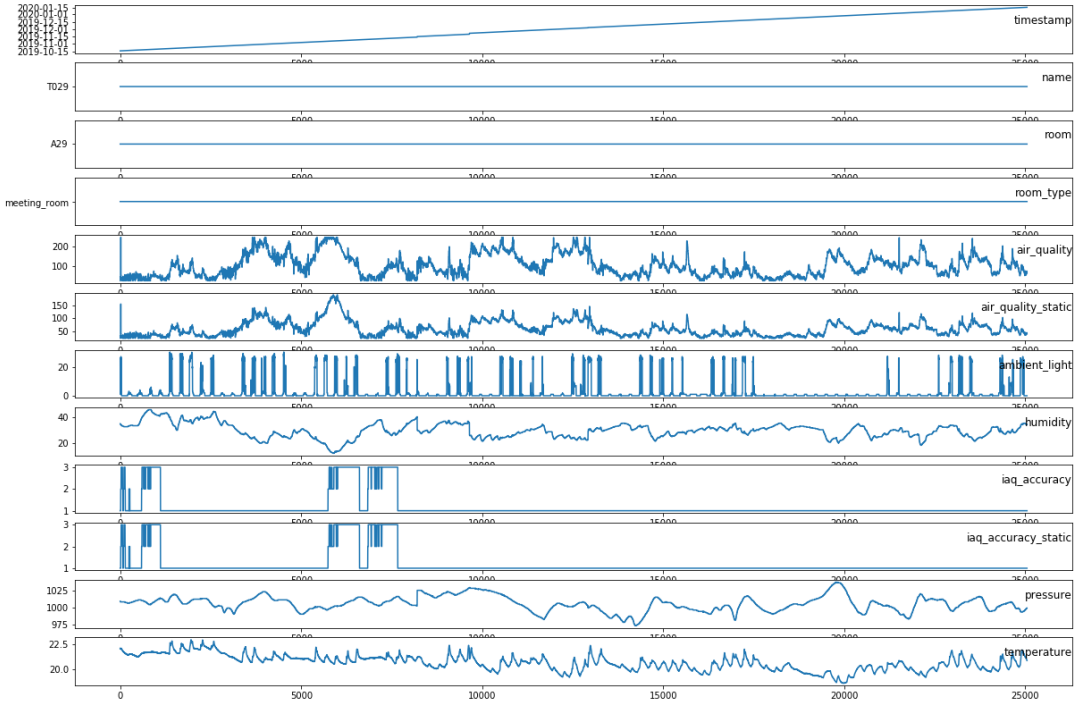

Figure 20. Timeseries data progression for room a29

The data is non-stationary, air quality and air quality static have an identical progression over time. Humidity, pressure and temperature are observed to be non-stationary and independent of each other. From Figure 13 we observe that air quality static has progressed over time and some anomalies and peaks have been noticed for the timeline of 25-10-2019 to 01-11-2019, 01-12-209 to 7-12-2019 and 28-12-2019 to 10-01-2020. These peaks in data are useful for our machine learning models to learn and predict. The average air quality in room_a29 is 61.40.

3. Room a30 - Meeting room

Like above rooms histograms for all columns in the data for meeting room room a30 are generated (in figure 21)

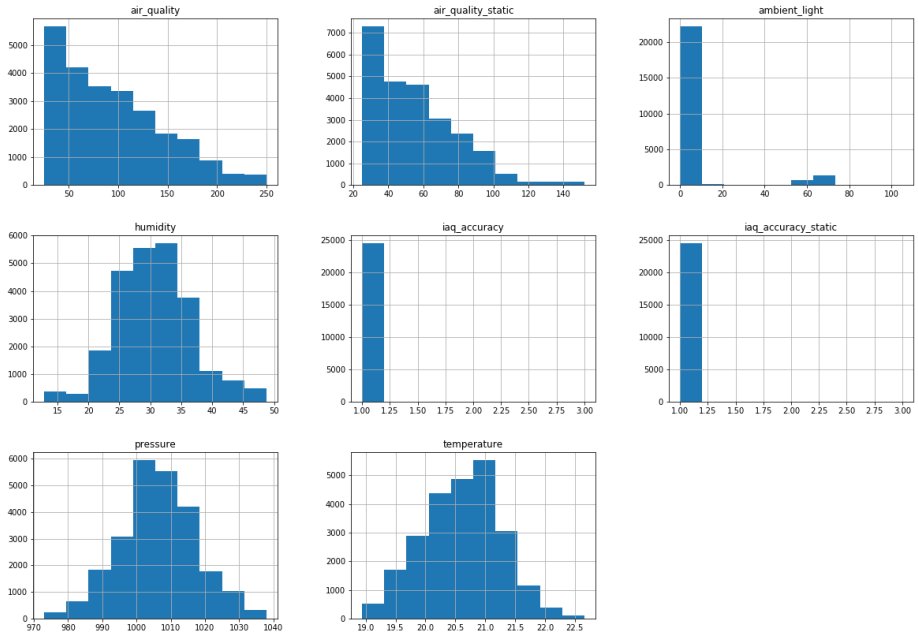

Figure 21. Emprical analysis for room a30

Air quality and air quality Static mostly happen to in good range (i.e.: 0-100) Some anomalies observed (ie: 150-250) on some occasions (Coway 2016). Ambient light is in two extremes either 1-9 or 60-80. Most of the humidity ranges from 25-40 grams per cubic meter which is healthy for humans. IAQ and iaq accuracy static are observed to be 1 for all instances. The pressure is mostly distributed in range 990 to 1020. Temperatures in the room are observed to be optimal mostly.

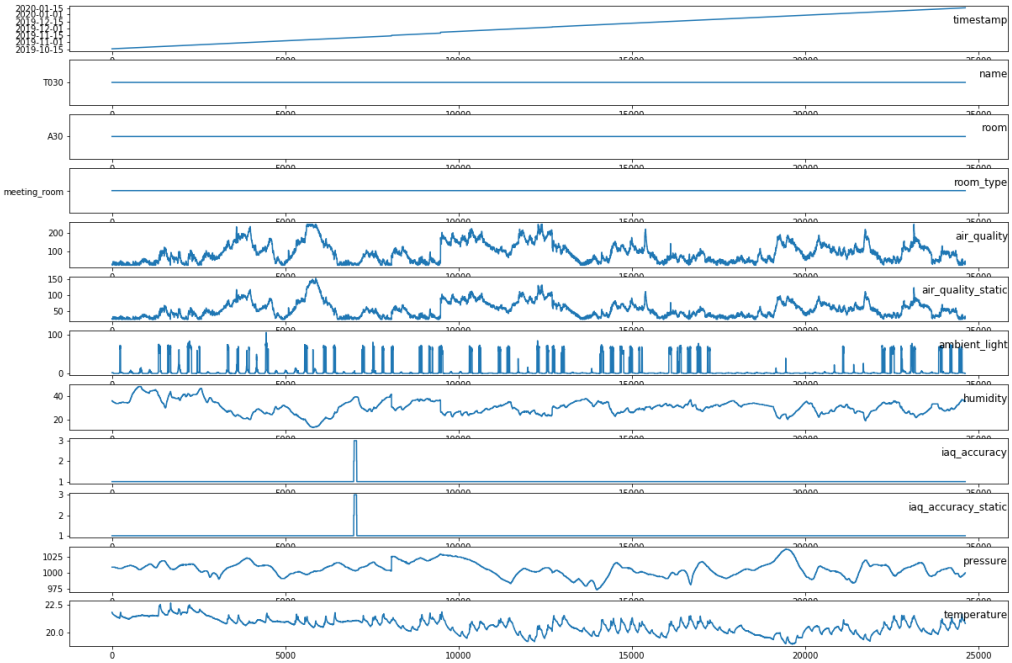

Figure 22. Timeseries data progression for room a30

In figure 22, a time-series sensor data for room_a30 data progression over time can be observed in these graphs. Some anomalies and peaks are noticeable for ambient light, IAQ accuracy and IAQ accuracy static. Air quality and air quality static have identical progression of data over time and the data is non-stationary. Likewise humidity, pressure and temperature are observed to be non-stationary and independent to each other. Some anomalies and peaks for air quality have been noticed in Figure 13 for timeline of 11-01-2019 to 12-11-2019 and 01-12-209 to 10-12-2019.

4.2.2 Feature Engineering

Feature engineering is transforming raw data into meaningful features so that data can be better represented and prepared for predictive modeling (Severyn & Moschitti 2013). As a result model accuracy is improved on unseen data. This section describes feature engineering on the data. Feature engineering steps involved feature extraction, correlation and scaling to prepare data for machine learning model training. Let us look into each step.

4.2.2.1 Feature Extraction After exploring data and identifying patterns in the above section, we clearly see important data parameters or columns that correlate to the air quality inside a room. Based on the data analysis, these are the parameters or columns that we will choose for training machine learning algorithms to predict air quality inside a room after 15 minutes.

1. air quality static
2. ambient light
3. humidity
4. iaq accuracy static
5. pressure
6. temperature

In order to predict air quality 15 minutes ahead, a new feature is created "future air quality" which is 15 minutes ahead of the current event, this feature is created by shifting the column "air quality static" three rows ahead. As each row or event in air quality static is created at 5 minutes interval, shifting it 3 rows ahead to create a new column will give us a column named "future air quality" which has 15 mins ahead air quality for given air quality static. After selecting needed columns and creating needed features, here is a snapshot of data in figure 23.

	timestamp	name	room	room_type	floor	air_quality	air_quality_static	ambient_light	humidity	iaq_accuracy	iaq_accuracy_static	pressure	temperature	future_air_quality
0	2019-10-16 11:47:25	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.82	27.0
1	2019-10-16 11:52:29	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.82	27.0
2	2019-10-16 11:57:41	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.80	27.0
3	2019-10-16 12:02:53	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.79	25.0
4	2019-10-16 12:07:57	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.78	25.0
5	2019-10-16 12:13:09	T030	A30	meeting_room	A	30.0	27.0	2.0	36.27	1.0	1.0	1009.0	21.78	25.0
6	2019-10-16 12:18:13	T030	A30	meeting_room	A	25.0	25.0	2.0	36.27	1.0	1.0	1009.0	21.76	25.0
7	2019-10-16 12:23:25	T030	A30	meeting_room	A	25.0	25.0	2.0	36.27	1.0	1.0	1009.0	21.75	25.0
8	2019-10-16 12:28:37	T030	A30	meeting_room	A	25.0	25.0	2.0	36.27	1.0	1.0	1009.0	21.73	25.0
9	2019-10-16 12:33:41	T030	A30	meeting_room	A	25.0	25.0	2.0	36.27	1.0	1.0	1009.0	21.73	27.0

Figure 23. Feature Engineering data snapshot

4.2.2.2 Feature Correlation Data and feature correlation is an important step in the feature selection for machine learning model training, especially when the data type for the features is continuous, as it is in our case. Pearson Correlation Coefficient can be used with continuous variables that have a linear relationship (Benesty et al. 2009). To understand the relationship, we observed data and feature correlation between the variable to predict and other attributes in the data. For the feature "future_air_quality" we calculated feature scores using Pearson correlation.

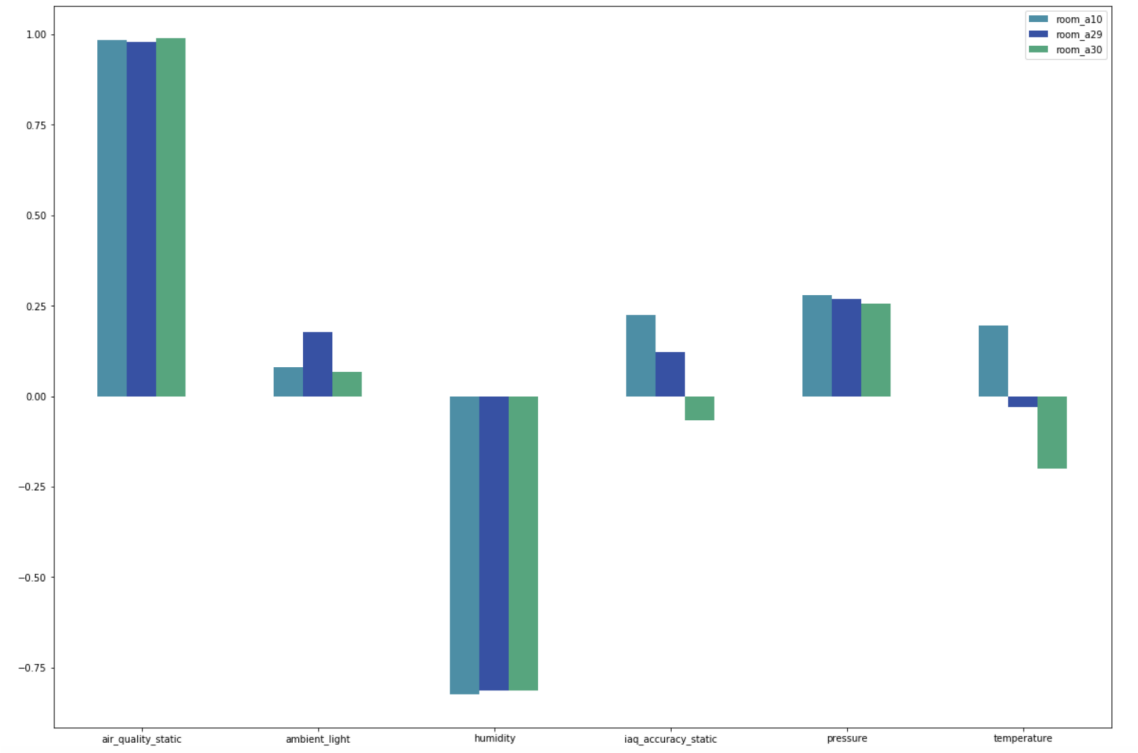

Figure 24. Feature correlation using pearson correlation.

We observed patterns for each room for our experiment as shown in figure 24. The feature "future_air_quality" shows a positive correlation with air_quality_static, pressure, ambient_light and iaq_accuracy to some extent. Positive correlation implies that feature A increases then feature B also increases or if feature A decreases then feature B also decreases. Both features have a linear relationship and move in tandem. In figure 24 we see a strong positive correlation for feature future_air_quality to air_quality_static and pressure. Humidity has a strong negative correlation which implies if feature A increases then feature B decreases and vice versa.

4.2.2.3 Feature Scaling The next step is to do feature scaling in order to get the data ready for Machine Learning training.

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

We perform standardization technique for feature scaling, in formal terms it is defined as

$$X_{new} = \frac{X_i - X_{mean}}{standardDeviation} \tag{7}$$

It re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1. With this we are ready for machine learning training with our new features and scaled data.

4.2.3 Model Training

In this section we will assess machine learning models trained to predict air quality based on variables we engineered in section 4.3. Below are the variables after feature engineering (that will be input to the model to predict future air quality).

- air quality static
- ambient light
- humidity
- iaq accuracy static
- pressure
- temperature
- future air quality

From the above variables we do **multivariate time series prediction** to predict future air quality 15 minutes after current time inside a particular room. These are the algorithms trained for data (section 4) for each room respectively (Each algorithm explained in detail in section 4.4)

1. Multiple Linear Regression (MLR)
2. Extreme Learning Machines (ELM)
3. Random forest Regressor (RFR)
4. Support Vector Regressor (SVR)

We performed cross validation using Timeseries Split as explained below to train and evaluate models.

4.2.3.1 Cross Validation - Timeseries Split In time series machine learning analysis, our observations are not independent but time dependent, so we cannot split the data randomly as we do in non-time-series analysis (Eg: Train, validation and test). Instead, we split observations along with the sequences.

Training data is split into multiple segments (10 segments for our experiment). We use the first segment to train the model with a set of hyper-parameter, to test it with the second. Then we train the model with first two chunks and measure it with the third part of the data. In this way we do k-1 times of cross-validation.

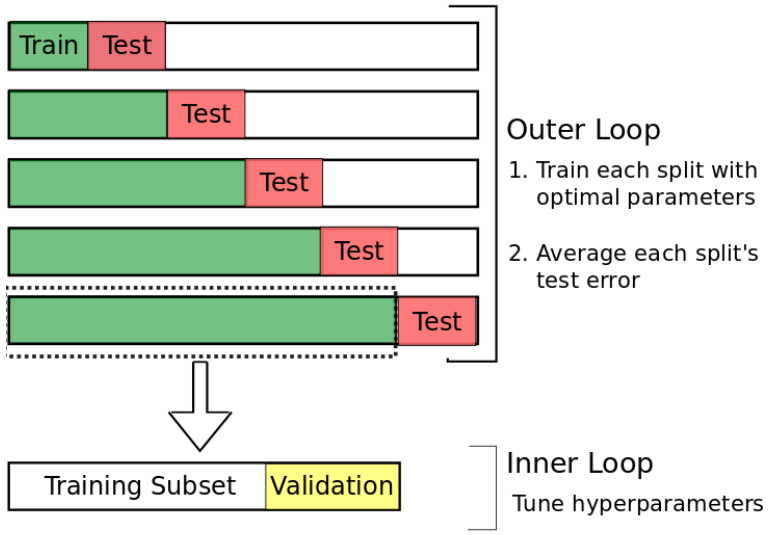

Figure 25. Timeseries split - Cross validation

For model training for our experiment timeseries split was implemented with 10 splits (using scikit-learn library).

4.2.4 Model Evaluation

We evaluate trained models using Timeseries split (cross validation) and root mean square error (RMSE) for metrics.

4.2.4.1 Metrics In order to assess model training performance **Root Mean Square Error (RMSE)** is used as it is a standard way to measure the error of a model in predicting quantitative data. Formally it is defined as,

$$RMSE = \sqrt{\frac{\sum(y_t - y_p)^2}{n}} \tag{8}$$

$y_p1, y_p2, y_p3.....y_pn$ are predicted values by the model. $y_t1, y_t2, y_t3.....y_tn$ are the obeserved values. n is the number of observations.

To calibrate final results for trained models performance Timeseries split (10 fold cross-validation) was implemented to take the average of RMSE of each fold. Also, trained models were tested on test data which is 20% of the total data for each respective room. Detailed results can be observed in table 2 for trained models on the data for respective rooms after Timeseries split (10-fold cross-validation), hyperparameter tuning and grid searching for best parameters for each algorithm.

After assessing the performance of each model on 10 fold cross-validation (Timeseries Split) and test data (20% of training data). Here is the ranking of model performance after model training in ascending order,

1. Multiple Linear Regression (MLR)
2. Support Vector Regressor (SVR)
3. Extreme Learning Machines (ELM)
4. Random forest Regressor (RFR)

Table 2. Model training results.

Model Training Results			
Room name	Algorithm	Cross Validation RMSE (train)	Test RMSE
Room A10	MLR	5.020	5.875
Room A10	ELM	6.325	6.208
Room A10	RFR	10.710	9.987
Room A10	SVR	6.046	5.977
Room A29	MLR	5.362	4.158
Room A29	ELM	11.202	4.223
Room A29	RFR	11.676	9.208
Room A29	SVR	8.073	4.176
Room A30	MLR	3.648	3.551
Room A30	ELM	7.920	3.895
Room A30	RFR	9.686	7.720
Room A30	SVR	5.177	3.55

4.2.5 Model Packaging

To do machine learning inference at the edge we have to serialize and package needed artifacts and machine learning models. Following are the artifacts serialized to be exported to production environments,

4.2.5.1 Input and output scaler : We performed a standardization technique for feature scaling. It re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1. Similarly, we have to scale incoming input data for model inference to be able to predict future air quality 15 minutes in the future. For this purpose, the feature scaling variable is serialized to a pickle file (.pkl).

4.2.5.2 Machine learning models : All trained and retrained ML models are serialized in the Open Neural Network Exchange (ONNX) format. ONNX is an open ecosystem for interoperable AI models, it enables model interoperability and serialization of ML and deep learning models in a standard format (.onnx). With this, all trained or retrained models and artifacts are ready to be exported and deployed to production environments.

4.3 Design Cycle: Proposed framework for Edge MLOps

In this section we design a framework for edge MLOps, we start by assessing For edge and cloud communication to be robust and realtime, it is essential to assess every ser-

vice provided by the cloud service to make an efficient synergy between edge and cloud. There is a range of services Azure offers to facilitate edge-cloud operations. We assessed some services on Microsoft Azure with a goal to facilitate continuous delivery, deployment and monitoring on edge devices, to orchestrate cloud to edge communication, data management, Machine Learning lifecycle management, federated learning, monitoring of Machine Learning models performance and edge devices. The maturity of these services has been assessed.

- Azure IoT Edge
- Azure IoT Central
- Azure DevOps
- Azure Machine Learning services
- Logic app
- Azure IoT Hub
- Azure Blob storage
- Azure data lake storage Gen2

After assessment, suitable services were chosen based on the maturity of the service for the experiment and efficiency for enabling orchestration and automated pipelines for synergy between edge and cloud. These services are discussed in section 4.3.1.

4.3.1 Azure Cloud Services used

In this section we reflect on the selected services for our experiments and the limitations of other services. Based on the assessment, these are the selected services,

- Azure ML Workspace: Automatically configure default storage, compute resource, and deployment target and integrate different services for example AKS, ADB.

- Azure DevOps for Source code managment, CI-CD pipeline job management and Native support for Azure ML workspace and other workspaces and services.

- Azure IoT Central: Enables bi-directional communication between IoT devices, edge devices and Azure. Central hub for IoT device and edge device management and fleet analytics.

- Azure blob storage: Storage for storing unstructured or structured data.

Azure IoT Edge was evaluated but as it was under development from Microsoft, we did not proceed with this service for our experiments. Some of the limitations were the integration of cloud to edge devices was restricted to C# and Microsoft native components, It did not have python sdk or python as the main language as it is a go-to language for Data Scientists and Machine Learning Engineers. These limitations made it less suitable for our experiments.

4.3.2 Continuous Integration for IoT to Edge

This section explains how we communicate with the IoT devices from edge devices and establish continuous integration between the respective IoT devices and the edge devices. Communication between IoT and Edge Devices is established using the MQTT protocol (as discussed in section 2.4). To fetch or collect sensor data at the edge device we use a communication protocol called MQTT protocol which is robust reliable and real-time (Hunkeler et al. 2008). It is a lightweight protocol based on a messaging technique with minimized data packets resulting in low network usage and latency. It is realtime and this makes it perfect for IoT applications.

4.3.3 Continuous Integration IoT to Cloud

In order to setup continuous integration for edge to cloud, there are some pre-requisites which need to be in place to configure CI-CD pipeline for edge-cloud. Here are the pre-requisites.

- Secure SSH access to running edge devices.
- Sensor to edge device(s) continuous integration working.
- Install needed packages in the edge devices (Eg: python packages, Azure ML).
- Monitoring script or process for Azure IoT central is running inside each edge device (for fleet analytics on the cloud).
- Docker installed in the edge devices.

Once we have done all the above steps, we proceed to configuring CI-CD for edge cloud. A process was implemented for Continuous Integration, delivery and deployment as shown in figure 26. These processes were setup to facilitate end to end continous integration from IoT devices to edge devices and edge devices to cloud.

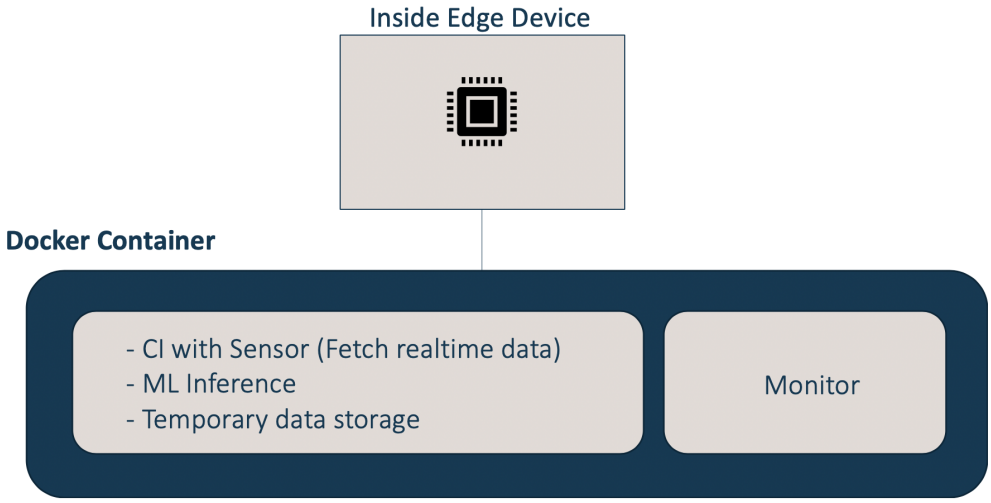

Figure 26. Docker container deployed in each edge device

There are two scripts or processes running inside docker container. These processes orchestrate data pipelines, machine learning, continuous integration and deployment.

- **Process 1**:
  - This process enables and maintains continuous integration of sensor to edge by fetching data in real time. This is achieved by subscribing to sensor topic using MQTT protocol. Upon receiving new data (which happens at an interval of every 5 minutes from a sensor), data is pre-processed by discarding or pruning unnecessary or extra data, cleaning and converting it to needed format for machine learning inference.
  - Machine Learning inference is done to predict future air quality in next 15 minutes on variables extracted from sensor data: air quality ambient light, humidity, iaq accuracy static, pressure and temperature. A Machine learning model previously trained in the cloud is deployed in the edge device inside the docker container machine.

– After getting a prediction using Machine learning model for data retrieved from sensor, both data from sensor and future air quality in next 15 minutes are concatenated together and appended to a csv file temporarily stored in the docker container.

- **Process 2**: For monitoring ML model performance at a set period of time everyday (time trigger) this process is triggered. Upon trigger it evaluates the machine learning model drift by evaluating the RMSE for future air quality predictions vs actual data. If the RMSE is greater than or equal to 8 it means model performance is poor, hence the process evokes a call to look for and deploy an alternative model from the ML model repository on the cloud.

These two processes running inside a docker container in each edge device ensure continuous integration for IoT devices to edge and applied machine learning to predict future air quality 15 minutes in the future for the incoming data from the sensor. This way of working is robust and scalable.

4.3.4 Fleet Analytics

Fleet analytics enables management, monitoring of edge devices (via telemetry data) and provides a holistic view of data collected from IoT devices data together with machine learning predictions.

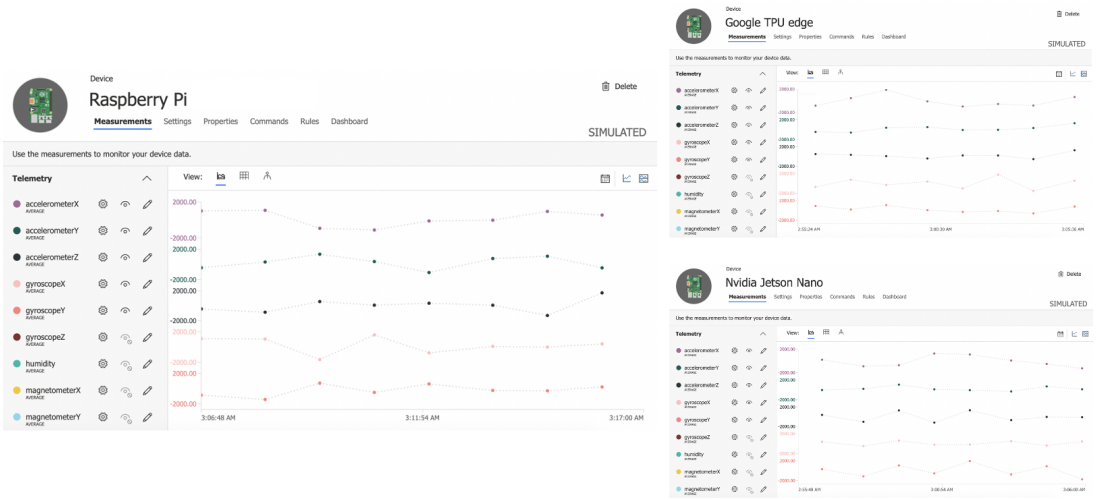

Figure 27. Fleet analytics for edge devices (telemetry data)

In figure 27 we observe fleet analytics for each edge device used in the experiment, it depicts device performance over a period of time with telemetry data like accelerometer, gyroscope, humidity, magnetometer, pressure and temperature. Useful information to monitor edge devices health and longevity. Data collected from IoT devices together with machine learning predictions are observed on a custom Power BI dashboard.

4.3.5 Continuous Delivery and Deployment for Edge

So far we have setup continuous integration for edge and cloud and this will be the driver for continuous delivery and deployment for machine learning models in the edge. In this section we will delve into important aspects of continuous delivery and deployment on the edge.

Figure 28 shows CI-CD pipeline setup on Azure DevOps. A preliminary step is required to configure this pipeline, that is to create a service connection for each edge device on Azure devops in order to connect to each device using secure ssh login via pipeline. This CI-CD pipeline orchestrates services used for the experiment on the cloud like Azure Machine Learning, Azure devOps and Azure Blob storage. Let us look into each phase in detail:

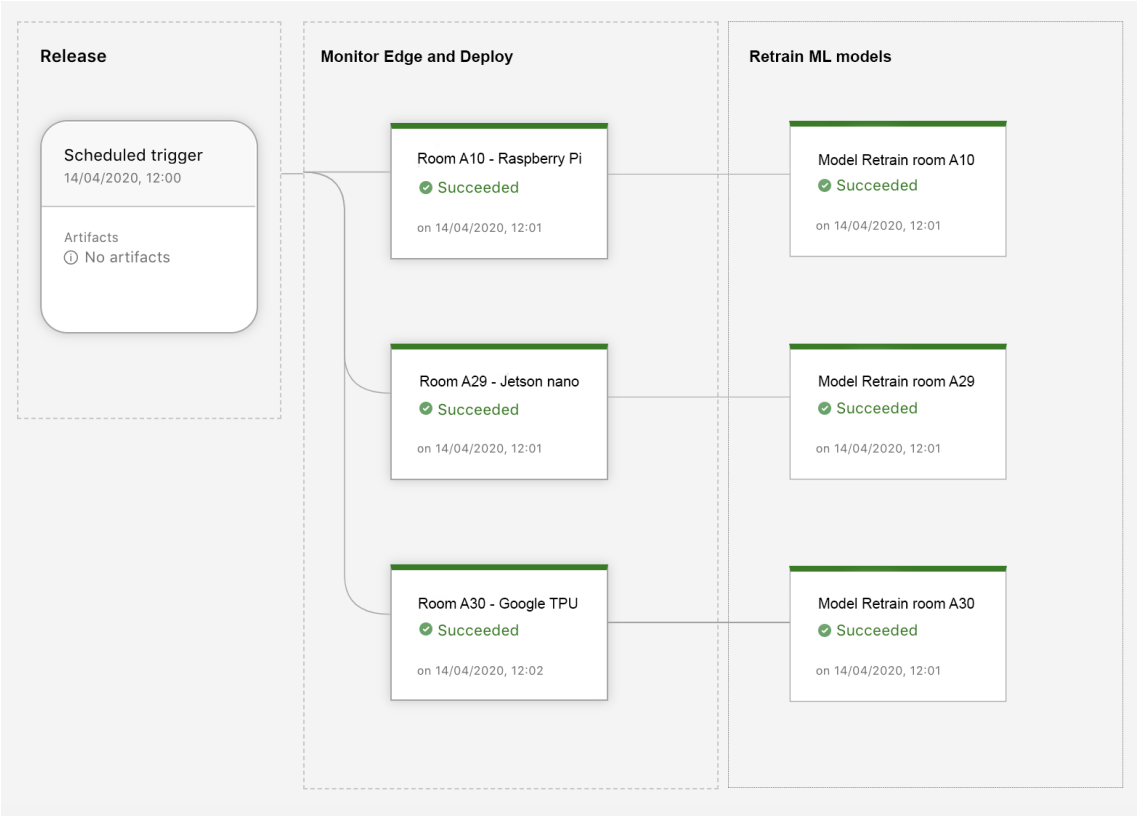

Figure 28. CI-CD pipeline for continuous delivery of ML models to the edge.

1. **Release**: A release is triggered on a set time every day to monitor edge devices to check model performance in realtime for model drift and if needed will deploy an alternate model if the model drift is high (Akkiraju et al. 2018). To finish the pipeline run a new model is trained upon the real-time data together with previously used training data. The trigger can be time-based or can be triggered manually on demand by an engineer. Other triggers are possible and optional like source code commit trigger and new model trained trigger, these triggers are not implemented in our experiment as there was no need.

2. **Monitor edge and deploy**: In this stage the focus is machine learning performance and Model monitoring, to evaluate the Model drift of models deployed in each edge device (Akkiraju et al. 2018). Parallel processes are run to access each edge device and access the CSV file which has recorded incoming data from sensors and predictions. From the CSV file we evaluate the future air quality (for the next 15 minutes) by comparing it to actual air quality recorded, for evaluation RMSE is used as the metric to assess the performance of the deployed model prediction in real-time. When RMSE is greater than or equal to 10 then the model is concluded to perform poorly, which results in evoking a call to replace the existing model with an alternative model from the ML model repository, this is done by deleting the existing model inside the docker container and replacing it with an alternatively selected model. This mechanism of model drift evaluation and model change ensures the continuous deployment of Machine learning models to the edge.

3. **Model Retrain**: This stage is run based in previous stage output. In the previous stage if an ML model is replaced with another one, then the replacement model is retrained on the cloud by the fine-tuning of the existing model with realtime data that it was used to infer. If performance is improved in terms of RMSE then the retrained model is stored in the ML model repository for future deployments. This ensures the models are updated with data drift in order to avoid higher model drifts in the real-time.

4.3.6 Proposed and implemented Architecture

In this section, each step of implemented architecture is discussed on a high level where as in previous sections we have seen the setup of continuous integration for IoT to edge devices and Continuous Delivery and Deployment for Edge. Below Figure 29 is implemented to enable automated machine learning at the edge, this end to end framework can be fully automated and run on autopilot. There are two main layers of the architecture - Edge inference and Cloud orchestration layer. To enable robust network communication between these two layers WLAN was used to implement continuous delivery triggers, send and receive data.

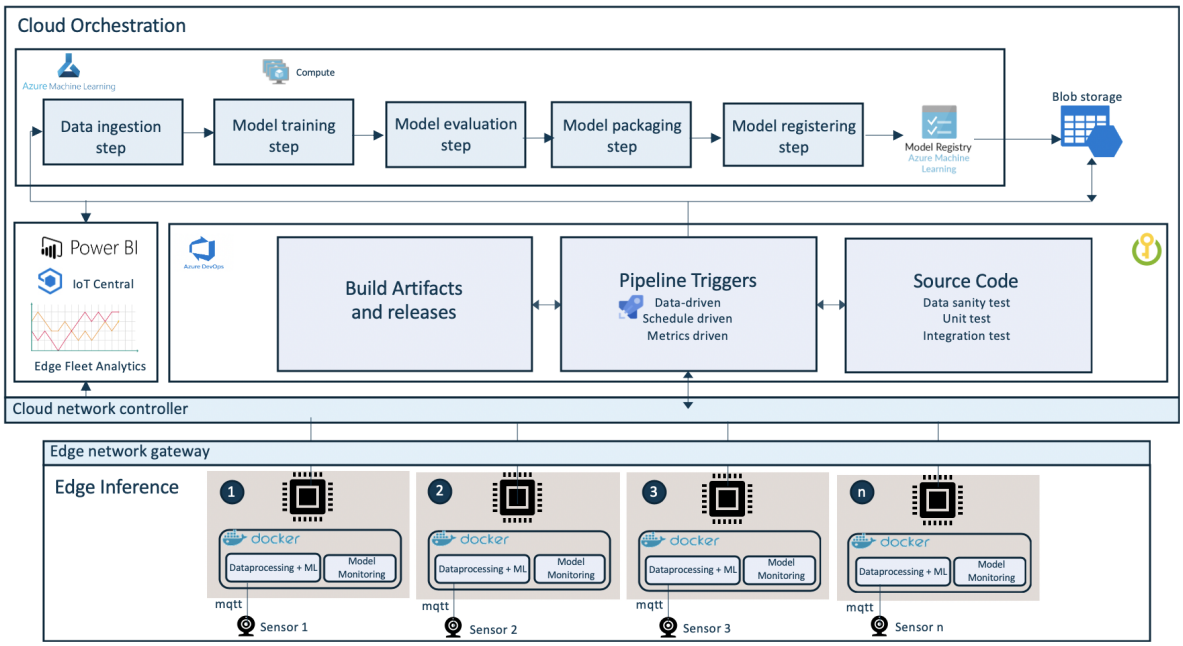

Figure 29. Proposed architecture

The architecture is made into two main modules or layers Cloud orchestration layer and Edge inference layer as shown in figure 29.

Cloud Orchestration

In Cloud Orchestration layer multiple services are running to perform parallel jobs mainly for four functions as described below,

1. **Machine Learning Pipelines**: This Machine learning pipeline is enabled by a service on Microsoft Azure called "Azure Machine Learning" service, it is an

enterprise-grade machine learning service to build and deploy models faster. It provides compute resources and data storage on demand to enable machine learning models training and is enabled by Jupyter notebooks or Databricks as a service to code, develop, and test machine learning models. It also comes with a machine learning model repository and container storage to enable faster deployment. All these features can be accessed by python SDK, which was done to implement our experiment. Here are the steps of the ML pipeline setup for the experiment,

- Data ingestion step: A python script that procures data needed for training and versions of the data used for machine learning model training. This way an experiment(model training) is audited and is back traceable.

- Model Training step: A python script performs data pre-processing, feature engineering, feature scaling before model training or retraining, and performs machine learning model training by optimizing the hyperparameters to train the best or optimal models.

- Model evaluation step: Once the ML model is trained this step makes sure to evaluate and test the model performance on test data using batch inference. The result of this is model accuracy and RMSE (for this experiment) on test data.

- Model packaging step: In this step, a trained and tested model is serialized in order to be exported to edge devices. ONNX serialization format was used to serialize and package trained and tested models.

- Finally the model is registered and stored in the model registry from where it is ready for quick deployments into edge devices.

2. **Storage**: As central storage, Azure blob storage is used. Blob storage allows Microsoft Azure to store arbitrarily large amounts of structured and unstructured data and serve them to users over HTTP and HTTPS. It can auto-scale as per the demand. Blob storage was used to store ML training data, sensors data, ML models, and telemetry data.

3. **Fleet Analytics**: Fleet analytics comprises of telemetry data from edge devices and IoT devices data together with machine learning predictions. Azure IoT central was used to have a central view of telemetric fleet analytics. For the experiment, Azure IoT central enabled telemetric fleet analytics for edge devices, health and performance could be monitored realtime on Azure IoT central and fleet analytics for IoT data plus machine learning prediction can be monitored on a custom Power BI dashboard.

4. **CI-CD**: Continuous integration and Continuous deployments enable continuous delivery to the edge layer. Azure DevOps service was used to maintain and version the source code used for model training, enable triggers to perform needed jobs in parallel, and to build artifacts and release for deployments to edge devices. Azure DevOps is the main driver for monitoring edge layer, Machine Learning models in production in the edge, and orchestration with other services on the cloud.

All these services and modules work in sync with each other to maintain and monitor machine learning models performance, maintain fleet analytics for edge devices, and to store and retrieve data as per the need. This layer is a foundation for the edge inference layer.

Edge inference

Edge inference layer focuses on orchestrating operations for IoT devices to edge devices, it also coordinates with the cloud orchestration layer to enable synergy between edge and cloud. In this layer edge operations with IoT devices, machine learning inference in realtime and synergy with cloud are performed.

1. Continuous integration and delivery for cloud to edge. Continuous integration is an important factor for automated systems, it requires high-quality development practice (edge cases proof) and robust process design, once implemented it is driven by high speed and low latency network for stable and robust communication and operations. In the experiment, a private network (WLAN) powered this communication between edge devices, sensors, and cloud layer, inside this network edge devices communicate and infer data from the sensors using MQTT protocol. Inside the edge device, all these operations

are run inside docker containers for stability and standardization. This way of operations ensures stable continuous delivery from cloud to edge and vice versa. Continuous delivery facilitates model deployments to the edge, data transfer, and monitoring (ML models and edge devices).

2. Automated Machine Learning at the edge Machine learning inference and monitoring are automated as part of continuous delivery and deployment operations orchestrated the services in the cloud orchestration layer. A periodic CI-CD trigger is implemented to evoke monitoring feature in the edge devices, to evaluate model drift and perform needed actions to replace the existing machine learning model with an alternative when needed. This way the whole process of machine learning inference at the edge is automated in realtime.

4.4 Empirical Cycle: AIoT Application

After the design cycle, we had a proposed framework for Edge MLOps for AIoT applications as discussed in section 4.3.6. In this section we implemented the framework to the problem context or AIoT application setup for our experiment. The machine learning models trained previously in section 4.2 were deployed in respective edge devices to predict future air quality in respective rooms. On every minute interval a machine learning inference was done for each edge device setup in respective rooms. During the experiment (12-03-2020 to 26-04-2020), 23 times new machine learning models have been changed or continuously deployed on respective edge devices. This was enabled by monitoring mechanism defined in the process, a trigger from Azure DevOps. Also 23 new models have been re-trained as a consequence of replacing a previously deployed machine learning model. For room a10 7 new ML models re-trained, room a29 7 new models re-trained and room a30 9 new models re-trained as shown in table 3.

Table 3. AIoT experiment machine learning inference results.

Realtime machine learning inference at the edge					
S.no	Date of model change	Edge Device	Deployed Model	Model Drift (RMSE)	Model Retrain (RMSE)
1	15-03-2020	Jetson nano 2	ELM	16.39	4.1
2	16-03-2020	Google TPU edge	RFR	14.23	6.3
3	16-03-2020	Raspberry pi 4	MLR	11.91	4.3
4	17-03-2020	Raspberry pi 4	ELM	13.27	8.1
5	22-03-2020	Jetson nano 2	SVR	22.32	6.2
6	24-03-2020	Google TPU edge	RFR	17.11	4.4
7	27-03-2020	Raspberry pi 4	MLR	16.22	4.7
8	29-03-2020	Jetson nano 2	ELM	30.28	8.2
9	30-03-2020	Google TPU edge	SVR	18.12	5.4
10	05-04-2020	Raspberry pi 4	MLR	12.92	3.2
11	10-04-2020	Jetson nano 2	SVR	17.21	5.2
12	11-04-2020	Google TPU edge	MLR	13.42	4.7
13	13-04-2020	Jetson nano 2	ELM	27.29	5.3
14	17-04-2020	Google TPU edge	RFR	17.46	6.9
15	19-04-2020	Raspberry pi 4	SVR	16.32	5.1
16	19-04-2020	Google TPU edge	MLR	11.91	3.4
17	21-04-2020	Jetson nano 2	ELM	23.26	7.3
18	22-04-2020	Google TPU edge	RFR	16.92	7.2
19	24-04-2020	Raspberry pi 4	SVR	17.87	5.2
20	25-04-2020	Google TPU edge	MLR	13.92	5.2
21	25-04-2020	Jetson nano 2	SVR	19.21	7.9
22	26-04-2020	Raspberry pi 4	ELM	23.57	6.4
23	26-04-2020	Google TPU edge	SVR	18.21	5.5

4.4.1 Observations

Here are the observations and learnings from the AIoT application experiments. Experiment was run for a month to assess robustness and scalability of the automated pipeline for edge AI.

1. CI-CD pipeline: During the experiment of 45 days, total of 45 time triggers were triggered, one trigger for each day at a set time of 12:00 EEST. And also 8 manual triggers were done to test the robustness of the pipeline. For the experiment all the triggers worked successfully and pipeline was executed each time. This assures the robustness of the CI-CD pipeline.

2. ML models deployed: Machine learning models were deployed 23 times for all 3 edge devices combined as a result of model drift greater than or equal to 10 RMSE. For room a10-raspberry pi had 7 new ML models deployed, room a29-jetson nano had 7 models deployed and room a30-google TPU had 9 models deployed during the experiment timeline.

3. ML models re-trained: Upon a new ML model deployed in the edge device, a new model is re-trained. 23 new models have been re-trained. For room a10 7 new ML models re-trained, room a29 7 new models re-trained and room a30 9 new models re-trained.

4. Edge vs Cloud comparison (Cost, energy and efficiency): We assessed cost, energy and operational efficiency of edge vs cloud machine learning inference based on our experiments. For reference edge device we used Raspberry pi 4 and for cloud compute a data science virtual machine DS2 v2 (Azure). We deployed machine learning models on both edge and cloud for machine learning inference in realtime. Table 4 shows results of monitoring the models deployed in 10 edge devices compared to 1 cloud node.

Table 4. Quantitative analysis - Edge vs cloud based on the experiments.

Edge vs Cloud inference based on the experiments		
	Edge devices (10)	Cloud node (1)
Device	Raspberry pi 4	DS2 v2 (Azure)
Computation	40 vCPUs (4x10)	2 vCPUs
RAM	40 GB (4x10)	7 GB
Temporary storage	640 GB (64 GB/device)	14 GB
Data pruned	22 %	0 %
ML inference/minute	1/device	10
Avg. inference time	0.2 seconds	2.2 Seconds
Total cost/month	$ 10/month	$ 93/month

Table 5. Quantitative analysis - Edge vs cloud scaled.

Edge vs Cloud inference - Scaled Scenario		
	Edge devices (1000)	Cloud node (100)
Device	Raspberry pi 4	DS2 v2 (Azure)
Computation	4000 vCPUs (4x10)	20 vCPUs (2x100 nodes)
RAM	4 TB (4x1000)	700 GB (7x100 nodes)
Temporary storage	64 TB (64 GB/device)	1.4 TB (14x100 nodes)
Data pruned	22 %	0 %
ML inference/minute	1/device	1000
Avg. inference time	0.2 seconds	2.8 Seconds
Total cost/month	$ 1000/month	$ 9300/month

In order to see a bigger picture and compare it to a real life production setup, we scaled it to 1000 edge devices and and 100 cloud nodes as shown in table 5.

4.4.2 Limitations

1. The proposed architecture is limited to cases where data privacy and privacy preservation is not essential for example in healthcare where patient data cannot be used as it is for model training due to laws like GDPR whose priority is to keep personnel's data private or anonymous. In these cases, federated learning approach can be implemented by extending the proposed framework as shown in figure 30:

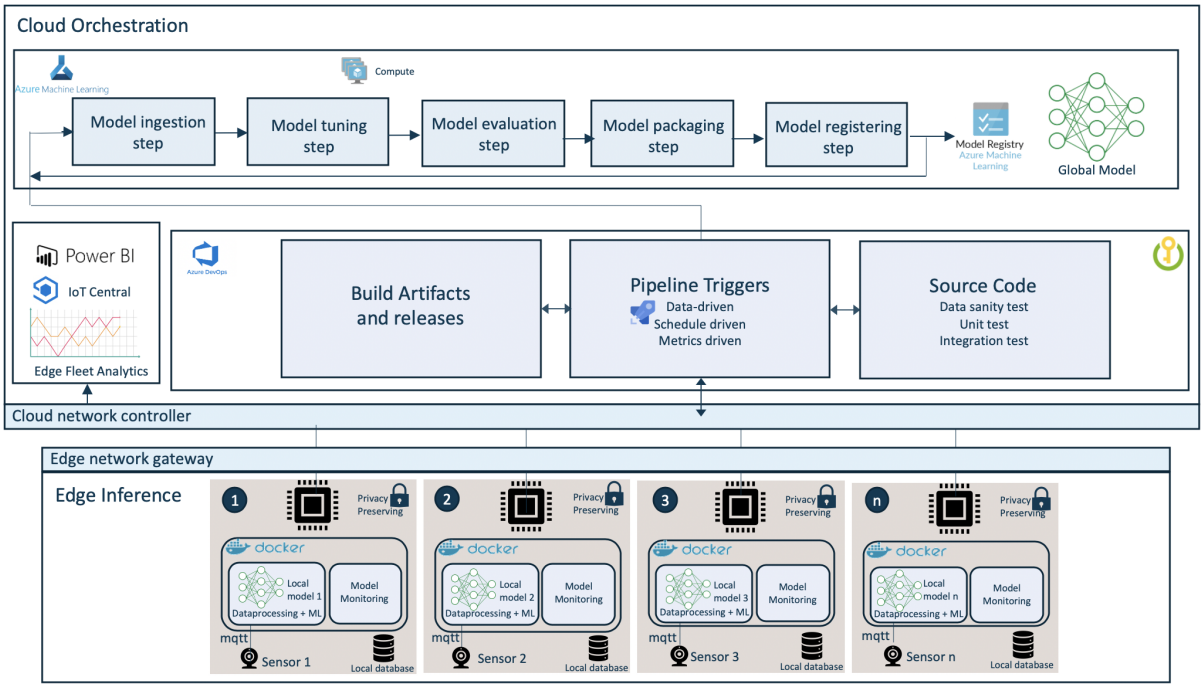

Figure 30. Extended Framework for Federated Learning

2. CI-CD parallel jobs: For each parallel job in CI-CD Pipelines in Azure DevOps, you can run a single job at a time which enables high scale parallel processing for edge-cloud operations. The current alternative is Azure app logic which enables parallel processing at scale and Azure IoT Edge which is under rapid development with python SDK and integration of CI-CD pipeline with Azure Machine Learning for continuous deployment of Machine Learning models at the edge.

5 RESULTS AND DISCUSSION

The results are based on the above experiments in section 4. Results are discussed in co-relation to the research methodology and goals we discussed in section 3. By iterative experimenting on the design cycle, a robust and scalable Edge AIOps framework for AIoT applications was curated and implemented using design science methodology described in section 3. Framework and architecture are discussed and implemented in section 4.3.6.

As the studies are of interdisciplinary nature, validation of design cycle was done in synergy with empirical cycle. Upon multiple iterations, here are the results of qualitative and quantitative analysis:

1. **Research problem analysis**: To investigate an improvement of problem in the field.

    - *Explore efficient ways of working in real-time with IoT data processing and machine learning*: End to end exploration was done in iterations to understand applied machine learning for AIoT applications. For efficient Networking (as discussed in section 2.4), Continuous integration (as discussed in section 4.3.3), Data pipelines, and Continuous deployment (as discussed in section 4.3.5) of machine learning models at the edge devices.

    - *To explore methods for applied machine learning for real-time multivariate time series forecasting*: A thorough investigation was done into applied machine learning techniques for multivariate time series analysis, some methods were applied in our experiments. Like Multiple Linear regression, Extreme Learning machines, Random forest regressor and support vector regressor (as discussed in section 3.2). Out of these methods Multiple linear regression performed the best in testing and real-time in production proceeded by support vector regressor.

    - *To observe how automated systems operate in real life and production settings*: After curating final architecture using design cycle iterations, we implemented a designed architecture for an automated system. Then we evaluated for 45 days (12-03-2020 to 26-04-2020) continuously and the running system was

monitored. Based on the observation, the CI/CD pipelines performed well with no instabilities or failures. The implemented architecture in the experiment setup was robust. We noticed 23 new models deployed over time upon 135 (45 x 3, 1 time trigger for every day for each device) CI/CD pipeline triggers and monitoring deployed models in production.

2. **Research design and inference design**: To survey possible methods.

   - *To assess the maturity of cloud services to enable operations on the edge and to identify the limitation*: In order to assess cloud service maturity on Microsoft Azure, we assessed each potential service one by one to see how it serves our experiments with respect to the software tooling (Python, Linux, Docker) selected for the experiments. We assessed Azure IoT hub, Azure IoT edge, Azure IoT central, Azure Machine Learning services, Azure DevOps, Azure Container instances, Azure blob storage and Azure data lake. After the assessment, we selected Azure Machine Learning Services (for data and machine learning pipelines), Azure DevOps (for CI/CD and source code management), Azure IoT central (for fleet analytics), Azure container instances (to manage docker containers) and Azure blob storage (for storage of pruned data). These services were selected based on compatibility with our software tooling, robustness, and scalability for our experiment.

   - *To curate a process for continuous delivery and deployment of machine learning models at the edge*: After final iterations for design and empirical cycles we have curated and implemented continuous delivery and deployment of machine learning models at the edge as described in sections 4.3.2 and 4.3.3. This end to end process for CI-CD for IoT to cloud worked and no interruptions were detected in the experiments, from fetching data from the IoT devices to performing machine learning predictions on the edge to deploying new models in the edge devices.

- *Machine Learning lifecycle management design for edge AI*: End to end pipelines were implemented for resource provisioning, data versioning, model Training and model storing for machine learning models deployed on the edge devices. Azure Machine Learning service was vital in orchestrating functionalities around the experiments the machine learning models as discussed in section 4.2.

3. **Validation of research and inference design**:

   - Robustness:

     – *Stability of CI-CD pipeline for edge to cloud*: Experiment was carried out for 45 days, every day at 12:00 a time trigger was executed to monitor the model drift of each model deployed in the respective edge devices and to retrain the machine learning models if needed. in total 135 time triggers were executed successfully without any failure. Also to test the robustness manual triggers from time to time, totally 27 manual triggers were executed and non of them failed to execute end to end. CI-CD pipeline was stable throughout the experiments.

     – *Stability of CI for IoT devices to edge devices*: Edge devices received data at every 5-minute interval from IoT devices without any failure.

     – *Machine learning models performance*: Everyday CI-CD pipeline would monitor the model drift of each model deployed in respective edge device, if the model drift (RMSE) of predictions vs actual data is more than 10 then a new model is retrieved and deployed in the respective edge device. During the experiment of 45 days, 23 new models were deployed based on the model drift metric RMSE crossing above 10.

     – *Machine Learning models retrained*: Whenever a new model is deployed in the edge device the previously deployed model is retrained or fine tuned with the real-time data collected during it's inference time period. As 23 new models were deployed during the experiment, 23 models have been retrained as well, all of them successfully without any machine learning

pipelines failures to retrain the models.

  - *Fleet Analytics*: fleet analytics for each edge device used in the experiment was collected for the duration of the experiment without any interruptions. Analytics for each device provided an overview of device performance over a period of time with telemetry data like accelerometer, gyroscope, humidity, magnetometer, pressure and temperature. Useful information to monitor edge devices health and longevity, all edge devices performance was stable overall.

  - *Data storage*: In total, 22 % incoming data from IoT devices was pruned (to send only essential data to cloud for storage). An aggregate of 38 MB of data was collected and stored on Azure blob storage without any interruptions or data leakage, this data comprised of sensor readings from IoT devices and machine learning model predictions. Data collection and storage pipeline worked and no interruptions were detected.

  - *Hardware*: All three edge devices (Raspberry pi4, NVIDIA Jetson nano 2, google TPU edge) used in the experiment performed without any interruptions and were stable for 45 days of experiment.

- Scalabilty: The proposed framework is scalable to multiple edge devices (depending on the use case) and is confined to the infrastructure (cloud and networking) and tools used to implement the framework and perform experiments.

- Application:

  - Application of the proposed framework is flexible for multiple industries like telecommunications, life sciences, energy, etc.

  - The proposed framework is also use case agnostic and can be applied to any AIoT application. In the case of privacy-preserving use cases, an extended framework proposed in section 4.4.2 can facilitate automated machine learning at the edge.

  - For federated learning based use cases an extended framework proposed in section 4.4.2 can be applied (Konečnỳ et al. 2016).

- Resources Optimization: Based on experiments the framework setup provided an almost 9-times improvement of resource use compared to the same experiment performed on cloud computing using micro services. These are the results of head to head comparision for edge computing vs cloud computing for the experiments,

  - There was an overall 9-times cost reduction.

  - 9-times increase in inference speed making the approach more suitable for real-time decision making using machine learning.

  - 22 % of unneccesary data was pruned and save on cloud reducing 22 % of storage costs.

Lastly, a perspective on hardware setup and installation for edge devices used in the experiments. Here are the ratings for each device in the table below. The hardware of each device used for the experiment is rated from 1 to 5 based on RAM, processor and hardware performance. Ease of use is about how easy was it to configure, install and run the device, it is rated from 1 to 5. 1 being hard and time-consuming and 5 being easy and time-efficient to get started and running.

Comparison				
Edge device	OS	Power usage	Hardware (1-5)	Ease of use (1-5)
Raspberry pi 4b	Raspbian OS	5V	5	4
Jetson Nano 2	Jetson nano developer kit OS	5V	5	3
Google TPU edge	Mendel OS	5V	4	2

6 CONCLUSION

This thesis has put forward a framework for edge computing, machine learning, cloud computing and low latency networks to work together to enable operational efficiency and create business value through resource optimization for real-time AIoT applications. The overall task has been to design a general framework or architecture that integrates continuous integration and continuous deployment of Machine learning models at the edge for AIoT applications.

In the introduction to this thesis, we look into the developments taken place over time in the infrastructure landscape with cloud computing and edge computing and the outlook towards using machine learning and low latency networks enabling real-time decisions at the edge, near the data origins. Following this, the research question was stated as:

RQ: *How can a framework that integrates continuous delivery and continuous deployment of machine learning models at the edge be implemented using state-of-the-art tools and methods?*

In order to address this objective, state of the art tools were selected based on market trends like popularity, adoption and readiness. Tools selected are commonly used software engineering tools by data scientists and machine learning engineers to build and deploy data and machine learning driven products and services. The tools used are Python, Linux OS, Docker for the software stack, and for infrastructure Microsoft Azure cloud was preferred based on popularity, adoption and readiness of the services.

This thesis is done in collaboration with partners, thanks to our industry and research partners TietoEvry and VTT Finland for provisioning experimenting facility in the 5G campus in Helsinki where rapid experiments were performed on three edge devices (Raspberry Pi 4, NVIDIA Jetson Nano 2 and Google TPU edge) located in three different rooms which had IoT devices setup to sense room conditions like temperature, pressure, ambient lighting and air quality. Machine learning algorithms were trained in a systematic approach (MLOps) as discussed in section 4.2. Multiple algorithms were trained to predict room air quality in the rooms 15 minutes in the future. These machine learning models were deployed and monitored in the edge devices during the experiment which lasted for 45 days.

To address our research question we followed a design science methodology proposed by Wieringa (2014). In an Iterative and structured approach, we implemented two cycles (Design cycle and Empirical cycle) to gain qualitative and quantitative results and conclusions for our design solution. After many iterations and optimizations throughout the experiments, we concluded with the proposed framework for Edge MLOps for AIoT applications as described in section 4.3.6. This solution has been concluded in section 5 to be robust, scalable and applicable across multiple industries for AIoT applications. However, it is needless to say that thesis has it's limitations. On the one hand, these may be related to simplifications for experiments like data modeling and pruning, limiting the scale of edge devices to three and perhaps some choices around limiting the tooling for the experiments, considering these factors some limitations are observed as follows,

1. The proposed architecture is Microsoft Azure cloud based. Needs to be generalized further.

2. The proposed architecture is limited to cases where data privacy and privacy preservation is not essential. To overcome this limitation an extended architecture was proposed in section 4.4.2, this extension enables asynchronous Federated Learning for data privacy and privacy preservation.

3. For each parallel job in CI-CD Pipelines in Azure DevOps, you can run a single job at a time which limits to a certain extent the ability to do parallel processing for edge-cloud operations, some other alternatives to azure for CI-CD pipelines are discussed in section 4.4.2.

On the other hand, these provide opportunities for further research as the following,

- Implementing this architecture in the industry and multiple sectors like life care sciences, energy, healthcare, etc.

- Exploring more methods for federated learning.

- Implementing the framework using a 5G network for more efficiency and faster operations at scale.

- Augmenting the Edge MLOps framework to fit other popular cloud infrastructure providers Google, Amazon and others to further generalize the framework.

With this thesis, we now better understand the benefits of applied AI on edge computing, with hands-on experience in designing and validating the proposed framework. As a result of this, we have a clear vision for further research.

REFERENCES

5G-Force. *5G-Force project, howpublished = https://5gtnf.fi/projects/5g-force/, note = Accessed: 2020-04-18.*

5GTNF. 2020, *5G Test Network Finland, howpublished = https://5gtnf.fi/overview/, note = Accessed: 2020-04-18.*

Akkiraju, Rama; Sinha, Vibha; Xu, Anbang; Mahmud, Jalal; Gundecha, Pritam; Liu, Zhe; Liu, Xiaotong & Schumacher, John. 2018, Characterizing machine learning process: A maturity framework, *arXiv preprint arXiv:1811.04871*.

Akusok, Anton; Björk, Kaj-Mikael; Miche, Yoan & Lendasse, Amaury. 2015, High-performance extreme learning machines: a complete toolbox for big data applications, *IEEE Access*, vol. 3, , pp. 1011–1025.

Beck, Michael Till; Werner, Martin; Feld, Sebastian & Schimper, S. 2014, Mobile edge computing: A taxonomy, In: *Proc. of the Sixth International Conference on Advances in Future Internet*, Citeseer, pp. 48–55.

Benesty, Jacob; Chen, Jingdong; Huang, Yiteng & Cohen, Israel. 2009, Pearson correlation coefficient, In: *Noise reduction in speech processing*, Springer, pp. 1–4.

Beyer, Betsy; Murphy, Niall Richard; Rensin, David K; Kawahara, Kent & Thorne, Stephen. 2018, *The site reliability workbook: Practical ways to implement SRE*, " O'Reilly Media, Inc.".

Bilal, Kashif; Khalid, Osman; Erbad, Aiman & Khan, Samee U. 2018, Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers, *Computer Networks*, vol. 130, , pp. 94–120.

Bonomi, Flavio; Milito, Rodolfo; Zhu, Jiang & Addepalli, Sateesh. 2012, Fog computing and its role in the internet of things, In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16.

Brian D. Noble, Dushyanth Narayanan James Eric Tilton Jason Flinn Kevin R. Walker School of Computer Science Carnegie Mellon University, M. Satyanarayanan. 1997, Agile Application-Aware Adaptation for Mobility, *Proceedings of the 16th ACM Symposium on Operating System Principles*.

Coway. 2016, *What is the Air Quality Index?, howpublished = https://www.cowaymega.com/air-quality-index/, note = Accessed: 2019-11-29*.

Dillon, Tharam; Wu, Chen & Chang, Elizabeth. 2010, Cloud computing: issues and challenges, In: *2010 24th IEEE international conference on advanced information networking and applications*, Ieee, pp. 27–33.

Drucker, Harris; Burges, Christopher JC; Kaufman, Linda; Smola, Alex J & Vapnik, Vladimir. 1997, Support vector regression machines, In: *Advances in neural information processing systems*, pp. 155–161.

Finland, Business. 2020, *Business Finland, howpublished = https://www.businessfinland.fi/en/do-business-with-finland/home/, note = Accessed: 2020-04-24*.

Girish Agarwal, hyperight, Ivana Kotorchevikj. 2019, *What is Edge AI and how it fills the cracks of IoT, howpublished = https://read.hyperight.com/what-is-edge-ai-and-how-it-fills-the-cracks-of-iot/, note = Accessed: 2019-12-21*.

Gunn, Steve R et al.. 1998, Support vector machines for classification and regression, *ISIS technical report*, vol. 14, no. 1, pp. 5–16.

Ha, Kiryong; Chen, Zhuo; Hu, Wenlu; Richter, Wolfgang; Pillai, Padmanabhan & Satyanarayanan, Mahadev. 2014, Towards wearable cognitive assistance, In: *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81.

Huang, Guang-Bin. 2014, An insight into extreme learning machines: random neurons, random features and kernels, *Cognitive Computation*, vol. 6, no. 3, pp. 376–390.

Huang, Guang-Bin. 2015, What are extreme learning machines? Filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle, *Cognitive Computation*, vol. 7, no. 3, pp. 263–278.

Huang, Guang-Bin; Zhou, Hongming; Ding, Xiaojian & Zhang, Rui. 2011, Extreme learning machine for regression and multiclass classification, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529.

Hunkeler, Urs; Truong, Hong Linh & Stanford-Clark, Andy. 2008, MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks, In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, IEEE, pp. 791–798.

Konečnỳ, Jakub; McMahan, H Brendan; Yu, Felix X; Richtárik, Peter; Suresh, Ananda Theertha & Bacon, Dave. 2016, Federated learning: Strategies for improving communication efficiency, *arXiv preprint arXiv:1610.05492*.

Liaw, Andy; Wiener, Matthew et al.. 2002, Classification and regression by randomForest, *R news*, vol. 2, no. 3, pp. 18–22.

Nelson, Patrick. 2016, *Just one autonomous car will use 4,000 GB of data/day, howpublished = https://www.networkworld.com/article/3147892/one-autonomous-car-will-use-4000-gb-of-dataday.htm note = Accessed: 2019-11-29*.

Popović, Krešimir & Hocenski, Željko. 2010, Cloud computing security issues and challenges, In: *The 33rd international convention mipro*, IEEE, pp. 344–349.

Preacher, Kristopher J; Curran, Patrick J & Bauer, Daniel J. 2006, Computational tools for probing interactions in multiple linear regression, multilevel modeling, and latent curve analysis, *Journal of educational and behavioral statistics*, vol. 31, no. 4, pp. 437–448.

Raj, Emmanuel. 2019a, *8 Enablers For Europe's Trustworthy Artificial Intelligence, howpublished = https://www.tietoevry.com/en/blog/2019/07/8-enablers-for-europes-trustworthy-artificial-intelligence/ note = Accessed: 2019-09-30*.

Raj, Emmanuel. 2019b, *Robust and scalable Machine Learning lifecycle for a high performing AI team, howpublished = https://www.tietoevry.com/en/blog/2019/12/robust-and-scalable-ml-lifecycle-for-a-high-performing-a note = Accessed: 2019-12-12*.

Ratkowsky, David A & Giles, David EA. 1990, *Handbook of nonlinear regression models*, 04; QA278. 2, R3., M. Dekker New York.

Ren, Ju; Guo, Hui; Xu, Chugui & Zhang, Yaoxue. 2017, Serving at the edge: A scalable IoT architecture based on transparent computing, *IEEE Network*, vol. 31, no. 5, pp. 96–105.

Rouse, Margaret. 2020, *Artificial Intelligence of Things(AIoT), howpublished = https://internetofthingsagenda.techtarget.com/definition/Artificial-Intelligence-of-Things-AIoT, note = Accessed: 2020-04-21*.

Satyanarayanan, Mahadev. 2001, Pervasive computing: Vision and challenges, *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17.

Satyanarayanan, Mahadev. 2017, The emergence of edge computing, *Computer*, vol. 50, no. 1, pp. 30–39.

Segal, Mark R. 2004, Machine learning benchmarks and random forest regression.

Severyn, Aliaksei & Moschitti, Alessandro. 2013, Automatic feature engineering for answer selection and extraction, In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 458–467.

Shahin, Mojtaba; Zahedi, Mansooreh; Babar, Muhammad Ali & Zhu, Liming. 2019, An empirical study of architecting for continuous delivery and deployment, *Empirical Software Engineering*, vol. 24, no. 3, pp. 1061–1108.

Shi, Weisong; Cao, Jie; Zhang, Quan; Li, Youhuizi & Xu, Lanyu. 2016, Edge computing: Vision and challenges, *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646.

Statista, Research Department. 2020, *Number of internet of things (IoT) connected devices worldwide in 2018, 2025 and 2030, howpublished = https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/, note = Accessed: 2020-05-15*.

Tan, Lu & Wang, Neng. 2010, Future internet: The internet of things, In: *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5, IEEE, pp. V5–376.

Techcrunch. 2016, *How AWS came to be, howpublished = https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/, note = Accessed: 2019-04-02*.

VTT. 2019, *Smart Otaniemi project, Platforms and Connectivity, howpublished = https://smartotaniemi.fi/pilots/platforms-connectivity/, note = Accessed: 2019-10-30*.

Westerlund, Magnus. 2018, A study of EU data protection regulation and appropriate security for digital services and platforms.

Wieringa, Roel J. 2014, *Design science methodology for information systems and software engineering*, Springer.

Wu, Yung Chang; Wu, Yenchun Jim & Wu, Shiann Ming. 2019, An outlook of a future smart city in Taiwan from post–Internet of things to artificial intelligence Internet of things, In: *Smart Cities: Issues and Challenges*, Elsevier, pp. 263–282.

Yi, Shanhe; Hao, Zijiang; Qin, Zhengrui & Li, Qun. 2015, Fog computing: Platform and applications, In: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, IEEE, pp. 73–78.

Zhang, Cha & Ma, Yunqian. 2012, *Ensemble machine learning: methods and applications*, Springer.

APPENDIX A

Edge devices setup and installation

This section describes the edge devices in detail. We discuss the process of setting up hardware and detailed steps to configure the edge devices for our experiment.

1. Raspberry Pi 4

Raspberry pi 4 is the latest small single-board computers developed by Raspberry Pi Foundation with below hardware specifications.

Specs

Processor	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory	4GB LPDDR4
Connectivity	wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 2.0 and 3.0 ports.
Internal Storage	None
MicroSD card	64GB
Power	5V DC via USB-C connector.
Size	88 x 58 x 19.5mm
OS	NOOBS - Raspbian OS (Linux based)

Peripherals

These are the needed peripherals in order to setup and configure raspberry pi 4, to get it up and running for our experiment.

- microSD card - 64 GB
- Micro-USB port for 5V power input or for data
- Gigabit Ethernet port
- HDMI output port
- DisplayPort connector

- USB keyboard and mouse

Once these peripherals are connected to the raspberry pi 4, we are set to configure software for our experiment as mentioned in the steps below.

1. Write Image to the microSD Card

   (a) A microSD card loaded with NOOBS, the software that installs the operating system is needed.

   (b) Downloaded Noobs OS from here: https://www.raspberrypi.org/downloads/noobs/

   (c) Write the image to your microSD card by using Etcher: https://www.balena.io/etcher/

2. First Boot and Setup

   (a) Keyboard and mouse settings:

   (b) Connecting to the internet

   (c) Installing software

   (d) Updating your Pi

   (e) Using the terminal

   (f) Install needed packages: Install needed python packages like numpy, pandas, sklearn, onnx and azureml.

   (g) Install Docker: Docker is a tool to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with required components such as libraries and other dependencies and deploy it as one package

   (h) Remote access: Setup remote access to login to the machine from a remote location. This will be essential to later facilitate edge device to cloud integrations.

Here is a reference to detailed steps followed above to setup raspberry pi 4 for our experiment. https://projects.raspberrypi.org/en/projects/raspberry-pi-using

2. Nvidia Jetson Nano 2 NVIDIA Jetson Nano enables the development of millions of new small, low-power AI systems. It opens new world of embedded IoT applications, including entry-level Network Video Recorders (NVRs), home robots, and intelligent gateways with full analytic capabilities.

Processor	128 CUDA core GPU, Quad-core ARM A57 processor
Memory	4 GB 64-bit LPDDR4
Connectivity	Wi-Fi requires external chip, USB 3.0, USB 2.0
Internal Storage	16 GB eMMC 5.1 Flash
MicroSD card	64GB
Power	Micro-USB port for 5V power input.
Size	69.6 mm x 45 mm
OS	Jetson nano developer kit OS (Linux based)

Peripherals

These are the needed peripherals in order to setup and configure Nvidia's Jetson Nano 2, to get it up and running for our experiment.

- microSD card - 64 GB
- Micro-USB port for 5V power input or for data
- Gigabit Ethernet port
- HDMI output port
- DisplayPort connector
- USB keyboard and mouse
- USB Wifi connector

Once these peripherals are connected to the Nvidia Jetson Nano 2, we are set to configure software for our experiment as mentioned in the steps below.

1. Write Image to the microSD Card

   (a) Download the Jetson Nano Developer Kit SD Card Image.

   (b) Write the image to your microSD card by using Etcher: https://www.balena.io/etcher/

2. First Boot and Setup

   (a) A green LED next to the Micro-USB connector will light as soon as the developer kit powers on. When you boot the first time, the Jetson Nano Developer Kit will take you through some initial setup, including:

      i. Review and accept NVIDIA Jetson software EULA

      ii. Select system language, keyboard layout, and time zone

      iii. Create username, password, and computer name

   (b) Using the terminal

   (c) Install needed packages: Install needed python packages like numpy, pandas, sklearn, onnx, azureml

   (d) Install Docker

   (e) Remote access: Setup remote access to login to the machine from a remote location. This will be essential to later facilitate edge device to cloud integrations.

Here is a reference to detailed steps followed above to setup raspberry pi 4 for our experiment. https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro

3. Google TPU edge Edge TPU is Google's purpose-built ASIC designed to run AI at the edge. It delivers high performance in a small physical and power footprint, enabling the deployment of high-accuracy AI at the edge.

Specs

Processor	Quad Cortex-A53, Cortex-M4F, GPU - GC7000 Lite, TPU coprocessor
Memory	1GB LPDDR4
Connectivity	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2
Internal Storage	8 GB eMMC
MicroSD card	64GB
Power	5V DC (USB Type-C)
Size	88 mm x 60 mm x 24mm
OS	Mendel OS (Linux based)

Peripherals and additional hardware for installation

These are the needed peripherals in order to setup and configure raspberry pi 4, to get it up and running for our experiment.

- microSD card - 64 GB
- USB-A to USB-micro-B cable (to connect your PC to the board's serial port)
- USB-A to USB-C cable (to connect your PC to the board's data port)
- 2 - 3A (5V) USB Type-C power supply (such as a phone charger)
- Ethernet cable or Wi-Fi connection
- USB keyboard and mouse

Unlike Raspberry pi 4 and Jetson Nano 2 this dev board needs to go through flashing and booting before it can run independently to perform the experiment. Once GoogleTPU edge is connected to a host computer(Linux or Mac) for flashing and booting we can start the process of booting the device and configuring the device to install needed software and packages for the experiment, here are the steps implemented in order to get GoogleTPU edge dev board ready for the experiment:

1. Install fastboot: Fastboot is basically a diagnostic tool used to modify the Android or linux file system from a computer when the smartphone or smart device is in bootloader mode.

2. Install Mendel Development tool (MDT): MDT is a command line tool that helps you perform tasks with connected Mendel devices, such as this GoogleTPU Dev Board. For example, MDT can list connected devices, install Debian packages on a device, open a shell with a device, and perform needed operations on the dev board. (installation done using python pip).

3. Flash the board's OS image: Some software especially firmware and OS is non-upgradable or non-rewritable while others are upgradeable, it is possible to install the firmware and OS of the device by connecting it to another computer (in the specified configuration) and then running the software provided by the manufacturer. This process is called flashing. takes about 5-7 minutes for flashing to complete for the dev board. When it's done, the board reboots.

4. Generate an SSH public/private key pair and setup remote access: push the key pair to the board's authorized keys file, which then allows you to authenticate with SSH. (Using MDT is just easier than manually copying the key over the serial console.)

5. Install needed packages: Install needed python packages like numpy, pandas, sklearn, onnx, azureml.

6. Install Docker.

Here is a reference to detailed steps listed by the Manufacturer, followed as above to setup Google TPU edge for our experiment:
https://coral.ai/docs/dev-board/get-started/#6-run-a-model-using-the-tensorflow-lite-api

APPENDIX B

Python libraries used to conduct the experiments,

- Numpy (version 1.17.0): NumPy is the fundamental package for array computing with Python and used in various computation operations.

- Pandas (version 0.25.0): It offer robust data structures for data analysis is and used for data handling and manipulation.

- Scikit-Learn (version 0.20.3): Implements various standard machine learning and model evaluation algorithms such as Support vector machines, ROC Curve, F1-Score etc and more.

- Matplotlib (version 3.1.1): Is a plotting package used for visualization of data.

- Seaborn (version 0.8.1): Is a statistical data visualization tool used fordata visualization.

- ONNX (version 1.2.0): ONNX is an open ecosystem for interoperable AI models, it enables model interoperability and serialization of ML and deep learning models in a standard format.

- AzureML (version 1.2.0): This module is python sdk for Azure Machine Learning services which enables data processing, data versioning, ML model training, packetizing, deploying and monitoring ML models.