National Technical University of Athens

School of Electrical and Computer Engineering

Division: Communication, Electronic and Information Engineering

**Edge Computing Real-Time Deployment of Anomaly Detection Machine Learning Algorithms**

# Diploma Thesis

Dimitriadis Michail Panagiotis

Supervisors: Dr. Theodora Varvarigou, Professor, ECE-NTUA professor

Dr. Chondrogiannis Efthymios, ICCS-NTUA Researcher

Dr. Lataniotis Christos, Irmos Technologies AG, CTO

Athens, February, 2024

National Technical University of Athens

School of Electrical and Computer Engineering

Division: Communication, Electronic and Information Engineering

**Edge Computing Real-Time Deployment of Anomaly Detection Machine Learning Algorithms**

# Diploma Thesis

Dimitriadis Michail Panagiotis

Supervisors: Dr. Theodora Varvarigou, Professor

Dr. Chondrogiannis Efthymios, ICCS-NTUA Researcher

Dr. Lataniotis Christos, Irmos Technologies AG, CTO

Approved by the three-member scientific committee on 15/02/2024

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Dr. Emmanouil Varvarigos | Dr. Symeon Papavassiliou | Dr. Theodora Varvarigou |
| Professor, ECE-NTUA | Professor, ECE-NTUA | Professor, ECE-NTUA |

Athens, February, 2024

.....

Dimitriadis Michail Panagiotis

**Graduate of School of Electrical and Computer Engineering, National Technical University of Athens**

# Περίληψη

Η αξιοπιστία των δεδομένων από αισθητήρες είναι κρίσιμη σε τομείς όπως η παρακολούθηση της υγείας κτιριακών κατασκευών. Με την πάροδο του χρόνου, οι αισθητήρες μπορεί να παρουσιάσουν δυσλειτουργίες, παράγοντας ανώμαλα δεδομένα που οδηγούν σε λανθασμένα συμπεράσματα. Η Μηχανική Μάθηση (ML) και ειδικότερα η Ανίχνευση Ανωμαλιών προσφέρει λύση, εντοπίζοντας ελαττώματα και διασφαλίζοντας την ποιότητα των δεδομένων. Η υλοποίηση τέτοιων αλγορίθμων σε απομακρυσμένα σημεία απαιτεί λύσεις edge computing. Αναπτύσσοντας ενσωματωμένα συστήματα ανά κτίριο σε τοπικό επίπεδο, οι αλγόριθμοι ML μπορούν να εκτελούνται επιτόπου, εντοπίζοντας ανωμαλίες τοπικά, μειώνοντας την ανάγκη για κεντρική επεξεργασία.

# Λέξεις Κλειδιά

Μηχανική Μάθηση, Ανίχνευση Ανωμαλιών, Συνεχής Ενσωμάτωση/Συνεχής Ανάπτυξη (CI/CD), Ανάπτυξη Ιστοσελίδων Full Stack, DevOps, Docker, Ανάπτυξη σε Κλίμακα, Portainer, Raspberry.

# Abstract

The reliability of sensor data is critical in fields such as structural health monitoring. Over time, sensors may malfunction, producing anomalous data that can lead to incorrect conclusions. Machine Learning (ML), and specifically Anomaly Detection, offers a solution by identifying faults and ensuring data quality. Implementing such algorithms in remote locations requires edge computing solutions. By deploying embedded systems locally at each building, ML algorithms can run on-site, detecting anomalies in real time and reducing the need for centralized processing.

# Keywords

Machine Learning, Anomaly Detection, Continuous Integration/ Continuous Deployment (CI/CD), Full Stack Web Development, DevOps, Docker, Deployment at Scale, Portainer, Raspberry

# Acknowledgements

I would like to extend my deepest gratitude to all those who have supported me throughout the development of this thesis.

First and foremost, I wish to thank my advisor **Dr. Christos Lataniotis**, CTO of **Irmos Technologies AG**, for their invaluable guidance, encouragement, and patience. Your expertise and feedback were crucial in shaping this work, and I am sincerely grateful for the time and effort you invested in helping me grow academically and professionally.

I would also like to thank **Dr. Efthimios Chondrogiannis** for his insightful advice and willingness to assist me during various stages of my research.

Thank you all for your support and contribution to this work.

# Contents

# Figures table

# Equations Table

# Εκτενή Περίληψη στα Ελληνικά

Η εξασφάλιση της αξιοπιστίας των δεδομένων από αισθητήρες είναι κρίσιμη σε τομείς όπως η παρακολούθηση της υγείας των κατασκευών, η περιβαλλοντική παρακολούθηση και η βιομηχανική αυτοματοποίηση. Με την πάροδο του χρόνου, οι αισθητήρες ενδέχεται να παρουσιάσουν δυσλειτουργίες ή να επηρεαστούν από εξωτερικές παρεμβολές, με αποτέλεσμα δεδομένα χαμηλής ποιότητας που εμποδίζουν την ακριβή ανάλυση και τη λήψη αποφάσεων. Στο πλαίσιο της παρακολούθησης της υγείας των κτιρίων, όπου οι μετρήσεις δονήσεων είναι κρίσιμες για την αξιολόγηση της σταθερότητας των κατασκευών, η ακεραιότητα των δεδομένων από τους αισθητήρες γίνεται εξαιρετικά σημαντική. Οι ελαττωματικοί αισθητήρες μπορούν να παράγουν ανώμαλα δεδομένα, οδηγώντας σε λανθασμένα συμπεράσματα σχετικά με την κατάσταση της κατασκευής.

Για την αντιμετώπιση αυτής της πρόκλησης, μια προσέγγιση βασισμένη σε δεδομένα που αξιοποιεί τη Μηχανική Μάθηση (ML), και συγκεκριμένα τους αλγορίθμους Ανίχνευσης Ανωμαλιών, είναι απαραίτητη. Αυτοί οι αλγόριθμοι μπορούν να εντοπίσουν ελαττωματικούς αισθητήρες διακρίνοντας ανώμαλα πρότυπα δεδομένων από την αναμενόμενη συμπεριφορά, εξασφαλίζοντας ότι μόνο αξιόπιστα δεδομένα χρησιμοποιούνται για περαιτέρω ανάλυση. Ωστόσο, η ανάπτυξη τέτοιων εξελιγμένων αλγορίθμων σε απομακρυσμένα σημεία παρακολούθησης δημιουργεί ζήτημα δυνατότητας διάθεσης.

Αυτό το πρόβλημα διάθεσης μπορεί να λυθεί αποτελεσματικά με τη χρήση Συσκευών Edge Computing. Με την ανάπτυξη μίας συσκευής ανά κτίριο, η επεξεργασία των δεδομένων μπορεί να γίνει τοπικά, με κάθε συσκευή να τρέχει τους αλγορίθμους Ανίχνευσης Ανωμαλιών βασισμένους στη ML. Αυτή η προσέγγιση εξασφαλίζει ότι μόνο τα ενδιαφέροντα και σχετικά δεδομένα μεταδίδονται στη κεντρική βάση δεδομένων για περαιτέρω ανάλυση, μειώνοντας σημαντικά τον όγκο δεδομένων που πρέπει να διαχειριστεί κεντρικά.

Συνεπώς, τα δύο κύρια σημεία που θα διερευνηθούν σε αυτή τη διατριβή είναι:

- Η διαχείριση πολλαπλών αναπτύξεων συσκευών edge computing σε μεγάλη κλίμακα.
- Η ενσωμάτωση αλγορίθμων Ανίχνευσης Ανωμαλιών ML σε κάθε συσκευή για έξυπνη ανίχνευση και απόρριψη ανώμαλων δεδομένων από ελαττωματικούς αισθητήρες, εξασφαλίζοντας την αξιοπιστία και την ακρίβεια των συλλεγόμενων δεδομένων.

# Chapter 1

# 1. Introduction

## 1.1. Objectives

The objective of the current thesis is to present different anomaly detection methods and algorithms, then compare the results of some of them, analyze the importance of the metrics used, and finally implement the most suitable of them to operate in real-time and in an on-line manner.

For the last task, the deliverable monitoring system will be using a wide range of technologies and techniques, in order to achieve all of the following live capabilities:

- Edge Computing Anomaly Detection (1, 2)
- Health monitoring of fleet deployed
- Over the air (OTA) reprogram of deployed fleets
- Anomaly report from sensor data to my custom application
- Display and browsing of anomaly reports for further study (Frontend)
- Application access control
- Security, Scalability, Maintainability, Efficiency

The following diagram shows an overview of the system to be created

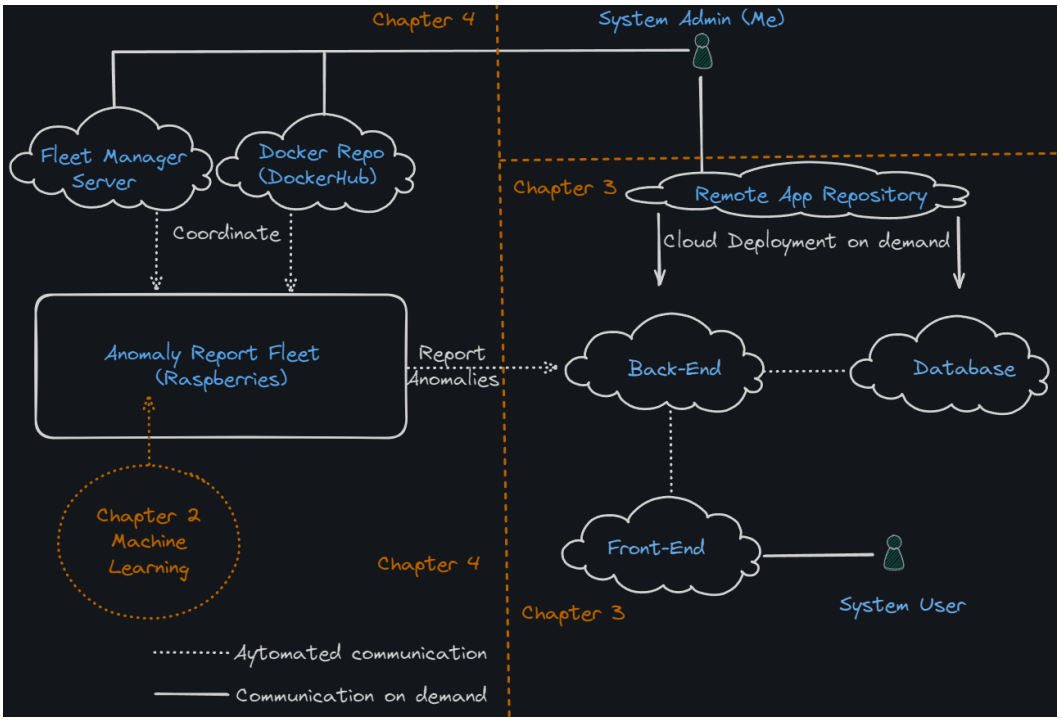


*Figure 1 System Overview*

## 1.2. Thesis Outline

In Chapter 2 we will define the fundamental terminology, conventions, as well as the classification of some of the most popular Machine Learning algorithms. We will also use 3 of them to test them against a common dataset and compare them using their key metrics. The best one will be used in my fleet. In Chapter 3 we will talk about the various components that a modern web application consists of, the way they interact with each other and in general the Full Stack Applications approach. Finally, in Chapter 4, we will show an overview of how edge computing devices work, why they play an important role in my system and how we will be able to control all my fleet easily and remotely.

# Chapter 2

# 2. Background

## 2.1. What is Machine Learning

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed. Learning algorithms in many applications that we make use of daily. These algorithms are used for various purposes like data mining, image processing, predictive analytics, etc. to name a few. The main advantage of machine learning is that we can produce algorithms that can perform potentially complex tasks without explicitly stating how to do so. Instead, they learn by examples (data-driven) through the so-called training process. (3)

The process of creating a mathematical model is called training, and the sample data used for this purpose are called training data. There are three primary Machine Learning paradigms, each with a distinct approach to "learning" or training: (4)

- Supervised Learning: Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labelled training data consisting of a set of training examples. The supervised machine learning algorithms are those algorithms which needs external assistance. The input dataset is divided into train and test dataset. The train dataset has output variable which needs to be predicted or classified. All algorithms learn some kind of patterns from the training dataset and apply them to the test dataset for prediction or classification. (3)

- Unsupervised Learning: These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data. The unsupervised learning algorithms learn few features from the data. When new data is introduced, it uses the previously learned features to recognize the class of the data. It is mainly used for clustering and feature reduction. (3)

- Semi Supervise Learning: Semi-supervised machine learning is a combination of supervised and unsupervised machine learning methods. It can be fruit-full in those areas of machine learning and data mining where the unlabeled data is already present and getting the labeled data is a tedious process. With more common supervised machine learning methods, you train a machine learning algorithm on a "labeled" dataset in which each record includes the outcome information. (5)

- Reinforcement Learning: Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. (3)

- Ensemble Learning: Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. Other applications of ensemble learning include assigning a confidence to the decision made by the model, selecting optimal features, data fusion, incremental learning, nonstationary learning and error-correcting. (6)

## 2.2. Anomaly Detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains. Of these, anomalies and outliers are two terms used most in the context of anomaly detection, sometimes interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities. (7)

The importance of anomaly detection is due to the fact that anomalies in data translate to significant (and often critical) actionable information in a wide variety of application domains. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination. An anomalous MRI image may indicate presence of malignant tumors. Anomalies in credit card transaction data could indicate credit card or identity theft or anomalous readings from a space craft sensor could signify a fault in some component of the space craft.

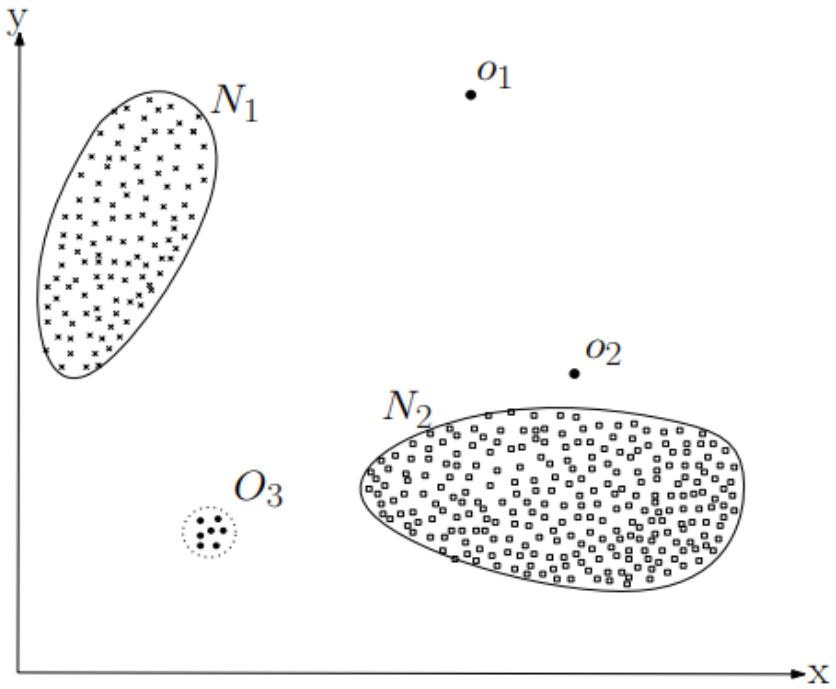


*Figure 2: Anomalies in a simple 2-dimensional data set illustration (8)*

The data has two normal regions, N1 and N2, since most observations lie in these two regions. Points that are sufficiently far away from the regions, e.g., points o1 and o2, and points in region O3, are assumed anomalies. Anomalies might be induced in the data for a variety of reasons, but all the reasons have a common characteristic that they are interesting to the analyst. In Figure 2, an example visualization of 2-dimensional dataset is shown. Points in different regions of the 2D space are denoted using different symbols, based on the group (a.k.a. cluster) that they belong to. (8)

## 2.3. Anomaly Detection Techniques

At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behavior. A straightforward anomaly detection approach, therefore, is to define a region representing normal behavior and declare any observation in the data which does not belong to this normal region, as an anomaly. But several factors make this apparently simple approach very challenging.

Defining a normal region which encompasses every possible normal behavior is very difficult. In addition, the boundary between normal and anomalous behavior is often not precise. Thus, an anomalous observation close to the boundary can be normal and vice-versa. When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the anomalous observations appear like normal, thereby making the task of defining normal behavior more difficult. In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future. The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while a similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus, applying a technique developed in one domain to another is not straightforward. Availability of labeled data for training/validation of models used by anomaly detection techniques is usually a major issue. Often the data contain noise which tends to be similar to the actual anomalies and hence is difficult to distinguish and remove. Due to the above challenges, the anomaly detection problem, in its most general form, is not easy to solve. In fact, most of the existing anomaly detection techniques solve a specific formulation of the problem. The formulation is induced by various factors such as nature of the data, availability of labeled data, type of anomalies to be detected, etc. Often, these factors are determined by the application domain in which the anomalies need to be detected. Researchers have categorized the anomaly detection techniques according to several major pattern approaches. (7) (9)

### 2.3.1. Classification Based (8)

Classification-based anomaly detection is a supervised method that involves training a classification model on labeled data, where anomalies are typically the minority class. The model learns to distinguish between normal and anomalous instances based on features extracted from the data. Once trained, the model can predict whether new instances are normal or anomalous based on their feature representation. Common classification algorithms used for anomaly detection include logistic regression, decision trees, support vector machines (SVM), and random forests. (8)

- **Support Vector Machine (SVM)**

A Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used primarily for classification tasks. It is especially effective in scenarios where data is not linearly separable and needs to be transformed into a higher-dimensional space for separation. SVMs are a type of supervised learning algorithm, meaning they require labeled data for training.

Some common uses and applications of the SVM algorithm include image classification, text classification and sentiment analysis, bioinformatics, handwriting recognition, face detection and recognition, anomaly detection, remote sensing, and satellite image analysis, as well as gesture recognition. The basic idea behind SVM is to find the optimal hyperplane that maximally separates the data into different classes, similarly to figure 2. To achieve that we follow these steps:

- **Data Preparation:** The dataset is split into training and testing sets and anomalies are labeled as such in the training set.
- **Training the Model / Find the Optimal Hyperplane**: The goal is to compute the optimal hyperplane that separates the normal and anomalous classes with the maximum margin. Mathematically, the hyperplane can be expressed as:

$$w \cdot x + b = 0$$

*Equation 1: Optimal Hyperplane*

Where w denotes the weight vector
$x$ is the input feature vector and
$b$ is the bias term.

- **Margin Maximization**: The margin is defined as the distance between the hyperplane and the nearest data points from either class, called support vectors. The objective is to maximize the margin, which is given by $2/|w|$
- **Formulating the Optimization Problem**: To maximize the margin, we need to solve the following optimization problem:

$$max\ (2/|w|\ )$$

*Equation 2 Optimization problem*

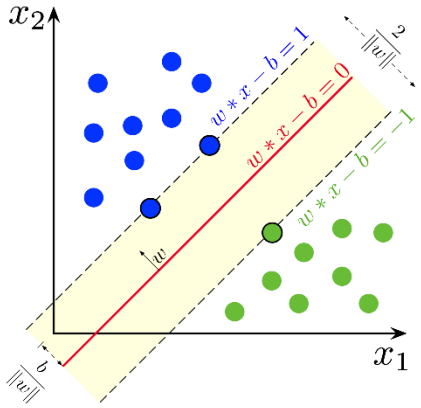


*Figure 3: support vector machines that define a splitting line among datapoints (10)*

The trained model is then used to predict the class labels of new data instances. Instances classified as anomalies are flagged for further investigation.

### 2.3.2. Tree based (11)

Tree-based machine learning algorithms are a powerful and versatile class of models used for both classification and regression tasks. Their defining characteristic is the use of **decision trees**, a flowchart-like structure where data is continuously split into subsets based on specific features. The primary strength of these models lies in their **interpretability**, as the decision-making process can be visualized and easily understood.

These algorithms mimic human decision-making processes, where a sequence of yes/no questions about the features leads to a final decision. The flow through the tree from the root to a leaf represents the decision-making path. Tree-based algorithms are widely used across industries, from finance to healthcare, because they handle diverse data types, including categorical and numerical data, and require little data preparation.

- **Isolation Forest**

Isolation Forest is an unsupervised, decision-tree-based algorithm originally developed for outlier detection in tabular data, despite often being classified under the umbrella of classification-based algorithms. The key principle behind the Isolation Forest algorithm is that anomalies, or outliers, are few and different in ways that make them easier to isolate from the majority of the data points. The algorithm works by recursively partitioning the dataset into smaller segments through a series of binary splits, with each split being selected randomly based on a feature and a split value.

For example, in a dataset containing annual incomes and total assets of individuals, the algorithm might randomly choose the feature "Net worth" and select a split value of, say, $2 billion. It then partitions the data based on whether an individual's worth is above or below this threshold. The process continues, with each tree randomly choosing features and split values, until all data points are isolated. Outliers, such as someone with an income of $10 billion, would be isolated faster because they differ more from the bulk of the population, whereas more common observations require more splits to be distinguished from the rest of the data.

The key steps in the Isolation Forest process can be summarized as:

- **Random Feature Selection**: A feature (e.g., "annual income" or "total assets") is chosen at random for splitting the data.
- **Random Split Value**: A random threshold value is chosen for the selected feature to divide the data into two branches.
- **Recursive Partitioning**: The process is repeated, recursively partitioning the data into smaller and smaller subsets.
- **Isolation of Data Points**: The recursion continues until each data point is isolated in its own partition, or until a stopping criterion is reached (e.g., the maximum depth of the tree).

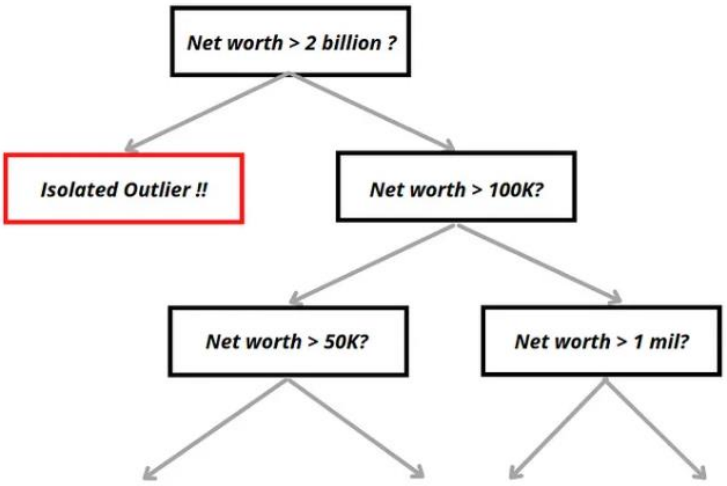


*Figure 4: Shows how a random tree isolates an Outlier (12)*

### 2.3.3. Clustering Based (8)

Clustering-based anomaly detection is an unsupervised method that involves grouping similar data points into clusters and identifying instances that are distant from their respective clusters. Anomalies are typically those data points that do not belong to any cluster or belong to sparse clusters. Common clustering algorithms used for anomaly detection include k-means, DBSCAN, and hierarchical clustering. (8) A presentation of one of the most simple algorithm follows:

- **DBSCAN**

It is a method that identifies distinctive clusters in the data, based on the key idea that a cluster is a group of high data point density, separated from other such clusters by regions of low data point density. The main idea is to find highly dense regions and consider them as one cluster. It can easily discover clusters of different shapes and sizes from a large amount of data, which contains noise and outliers.

The DBSCAN algorithm uses two major parameters:

minPts: The *minimum number of points* (a threshold) clustered together for a region to be considered dense i.e., the minimum number of data points that can form a cluster

eps (ε): A *distance* measure that will be used to locate the points in the neighborhood of any point.

The algorithm takes care of three concepts called **Core Points, Density Reachability** and **Density Connectivity.**

- **Core Points**: A point p is a core point if within its eps neighborhood, there are at least minPts points. Formally,

$$|N\_\varepsilon\ (p)| \geq min\ \ P\ ts$$

*Equation 3: Minimum points to be considered core point*

where Nϵ (p) denotes the number of points within eps neighborhood of p

- **Density Reachability:** A point $p$ is density reachable from a point $q$ if:

$$dist(p,q) \leq \varepsilon\ and\ |N\_\varepsilon\ (p)| \geq min\ \ P\ ts$$

*Equation 4: Density reachability*

- **Density Connectivity**: Points $p$ and $q$ are density connected if there exists a sequence of points $p\_1, p\_2, \dots, p\_n$ such that:
-

$$p\_1\ =\ p,\ p\_n\ =\ q,\ and\ \forall i\ \in\ \{1,2,\dots,n-1\},\ dist(p\_i, p\_(i+1)\ ) \leq \varepsilon$$

*Equation 5:Density Connectivity*

Additionally, $p$ and $q$ must be density reachable from some core point $o$.

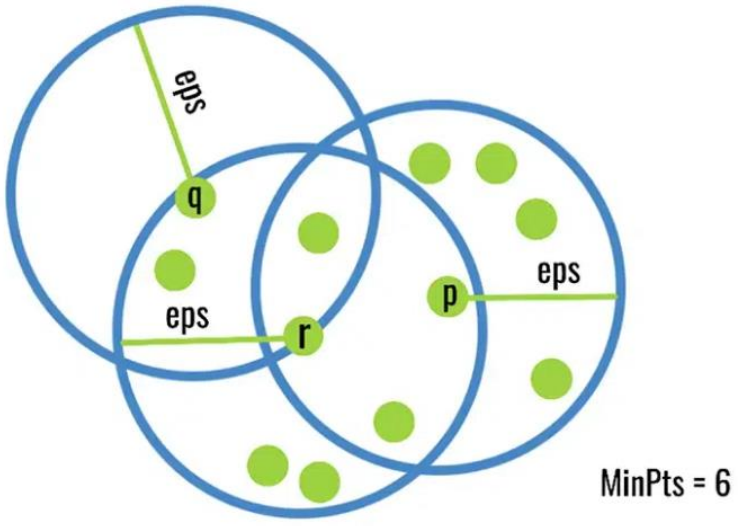


*Figure 5 : p and q are Density Reachable (13)*

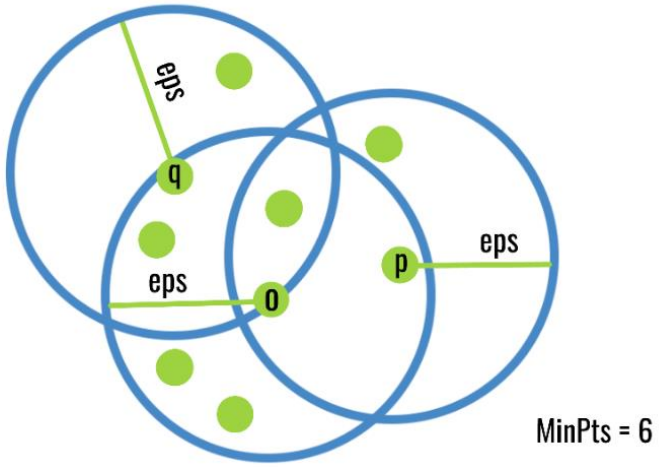


*Figure 6: Here p and q are Density Connected (13)*

In this method, there are three different types of data points:

**Core data point:** A data point which has at least 'minPts' within the distance of 'ε'.

**Border data point:** A data point which is in within 'ε' distance from core data point but not a core point.

**Noise data point:** A data point which is neither core nor border data point

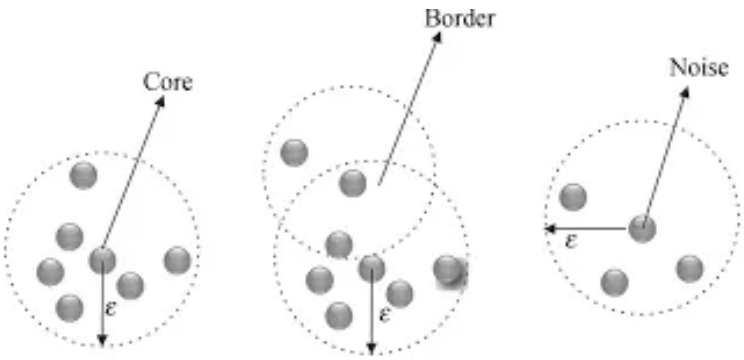


*Figure 7: Core, Border and Noise points visualized (14)*

Algorithmic steps for DBSCAN clustering

Initially, it starts with a random unvisited starting data point. All points within a distance 'ε' classify as neighborhood points.

It needs a minimum number of ‘minPts’ points within the neighborhood to start the clustering process. Otherwise, the point gets labeled as ‘Noise.’

All points within the distance ‘ε’ become part of the same cluster. Repeat the procedure for all the new points added to the cluster group. Continue till it visits and labels each point within the ‘ε’ neighborhood of the cluster.

On completion of the process, it starts again with a new unvisited point thereby leading to the discovery of more clusters or noise. At the end of the process, you ensure that you mark each point as either cluster or noise.

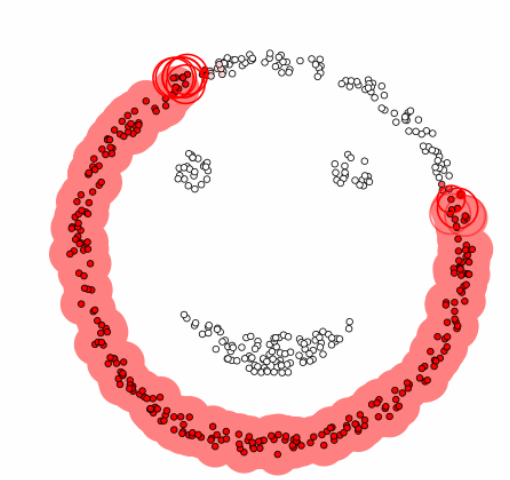

*Figure 8: A visual representation of how DBScan works (15)*

### 2.3.4. Nearest Neighbor Based (8)

- **K-nearest neighbors (KNN)**

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of ‘K’ training data and the class that holds the highest probability will be selected. In the case of regression, the value is the mean of the ‘K’ selected training points. Let's see the below example to make it a better understanding. (8)

Unlike many other machine learning algorithms, KNN does not create an explicit model or learn patterns from the data. Instead, it simply stores all the training data and makes predictions by comparing the test data to the stored points during inference. This is why KNN is often referred to as a lazy learner. KNN makes predictions by using the entire training dataset at prediction time. This can be memory-intensive because it requires storing all the data points and computing distances for each new prediction.

As a result, KNN can be computationally expensive for large datasets because it requires calculating distances between the test data and all training points. As the size of the dataset increases, the prediction time also increases significantly. In high-dimensional spaces, KNN can struggle because distance measures become less meaningful, a problem known as the curse of dimensionality. This can degrade the accuracy of the algorithm.

To address issues with large datasets, techniques such as data sampling, dimensionality reduction (e.g., PCA), or approximate nearest neighbor search can be used to reduce the dataset size while still maintaining the accuracy of KNN predictions. (16)

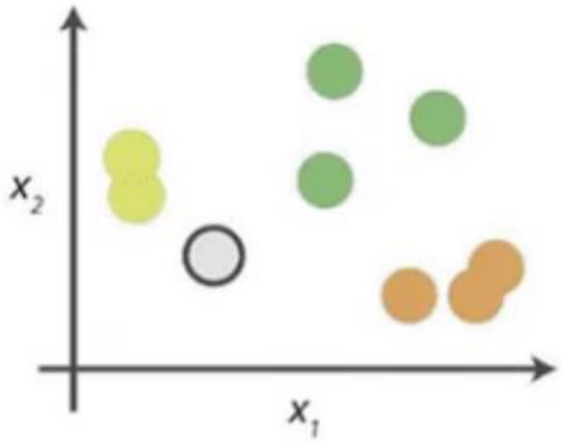


*Figure 9: 1st step of K-nearest neighbors (17)*

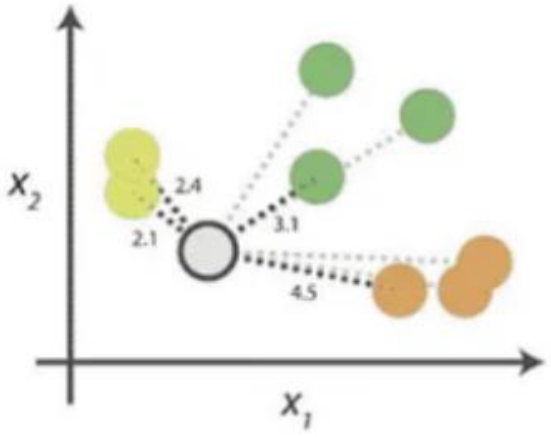


*Figure 10: 2nd step of K-nearest neighbors (17)*


*Figure 11: 3rd step of K-nearest neighbors (17)*

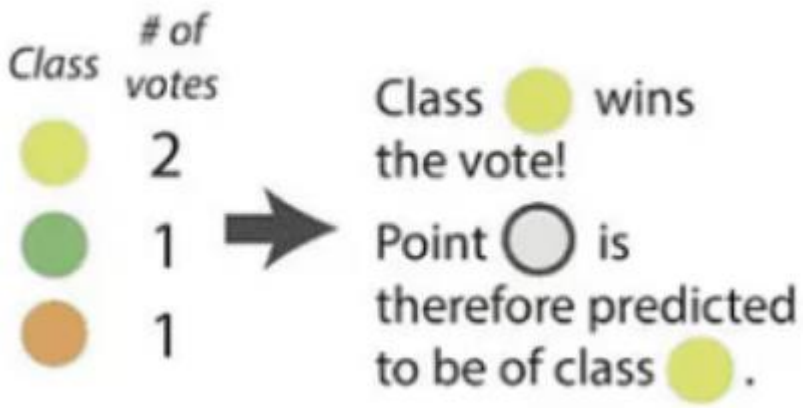


*Figure 12: 4th step of K-nearest neighbors (17)*

In Figure 9 we want to classify the gray point. In Figure 10 we measure the distances to the closest neighbors. In the next Figure we sort the neighbors from the closest to the furthest and finally we choose the right class for out gray point

- **Local Outlier Factor (LOF)**

Local Outlier Factor (LOF) is another popular anomaly detection algorithm, particularly useful for identifying outliers in a dataset, taking into advantage the K-nearest neighbors algorithm. LOF works by measuring the local density deviation of a given data point with respect to its neighbors. It then assigns an anomaly score to each data point based on how isolated the point is from its surrounding neighborhood. The key idea is that outliers are points that have a substantially lower density than their neighbors. These are the steps that are typically used

**STEP 1: Determine the Neighborhood:** For each data point in our dataset D, identify its k-nearest neighbors. The parameter k is typically chosen by the user and determines the size of the neighborhood. the k-distance of p is defined as:

$$d\_k\ (p) = dis\ tan\ \ c\ e(p, k - th\ nearest\ neighbor\ of\ p)$$

The k-distance neighborhood of a point p, denoted as $N\_k\ (p)$, includes all points whose distance to p is less than or equal to the k-distance of p:

$$N\_k\ (p)\ =\ \{dis\ tan\ \ c\ e(p, q) \leq d\_k\ (p)\}$$

*Equation 6: Determining neighborhood*

Where $q\ \in\ D$

**STEP 2: Calculate Local Reachability Density (LRD):**

Reachability Distance: For a point q and one of its neighbors o, the reachability distance of q with respect to o is the maximum of the Euclidean distance between q and o and the distance to the k-th nearest neighbor of o.

$$reach - dist\_k\ (p, o)\ =\ max\ \{d\_k\ (o), dis\ tan\ \ c\ e(p, o)\}$$

*Equation 7: Calculating reachability distance*

Local Reachability Density (LRD): This is the inverse of the average reachability distance of the point from its k-nearest neighbors. Points in dense regions will have high LRD values, whereas points in sparse regions (potential outliers) will have low LRD values.

$$lrd\_k\ (p)\ =\ ((\sum\_(\ \_(o \in N\_k\ (p))\text{^}\ )\text{▒}reach - dist\_k\ (p, o))/|N\_k\ (p)|\ )\text{^}(-1)$$

*Equation 8: Calculating LRD*

**STEP 3: Compute LOF Score:** The LOF score for a point is the average ratio of the LRD of the point's neighbors to the LRD of the point itself. If the LRD of a point is significantly lower than the LRDs of its neighbors, it is considered an outlier.

$$LOF\_k\ (p)\ =\ (\sum\_(\ \_(o \in N\_k\ (p))\text{^}\ )\text{^}\ \text{▒}\ 〚lrd\_k\ (o)\ 〛\ \backslash/lrd\_k\ (p))/|N\_k\ (p)|$$

*Equation 9:Calculating LOF*

### 2.3.5. Statistical Based (8)

Statistical-based anomaly detection involves modeling the statistical properties of the data and identifying instances that deviate significantly from these properties. Common statistical techniques used include Gaussian distribution modeling, z-score calculation, and hypothesis testing. (8)

- **z-score calculation**

A Z-score or standard score describes the measurement of distance between a data point and the mean using standard deviations.

**The mean**, often denoted by the **Greek letter μ (mu)**, is a measure of central tendency in a dataset. You may often heard as, the mean represents the “average” value of a set of numbers.

The standard deviation, often represented by the Greek letter σ (sigma), quantifies the amount of variation or dispersion in a dataset. In other words, it tells us how spread out the data points are from the mean. A smaller standard deviation indicates that the data points are close to the mean, while a larger standard deviation suggests greater variability.

$$\sigma = \sqrt{((\sum\_ \text{^ ▒}(x\_i - \mu)\text{^}2\ )/N)}$$

*Equation 10: Standard deviation (σ)*

Where:

σ = standard deviation

N = size of population

Xi = each value from population

μ = the mean

The formula to calculate the Z-score for a data point (X) in a dataset with a mean (μ) and standard deviation (σ) is as follows:

$$z = (x — \mu) / \sigma$$

*Equation 11: z-score calculation*

In simple terms, the Z-score is the result of subtracting the mean from the data point and then dividing the difference by the standard deviation. Datapoints that have Z-score above 3 (meaning the are more the 3 σ from the mean) are considered outliers.

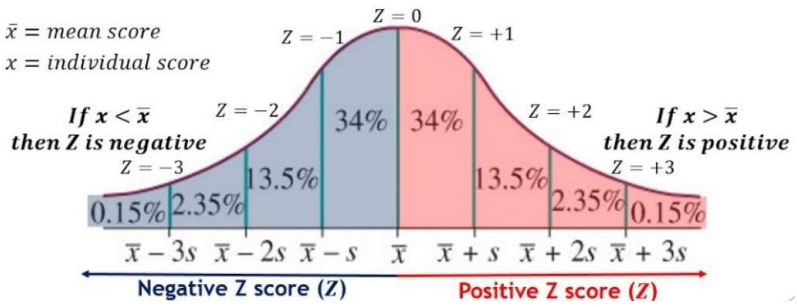


*Figure 13: Z-score distribution diagram (18)*

### 2.3.6. Information Theoretic Based (8)

Information-theoretic anomaly detection is rooted in the principles of information theory, where data points are evaluated based on the amount of information they carry relative to others in a dataset. This method seeks to quantify and analyze the informational content of each point to identify anomalies that deviate from the expected informational patterns. Techniques such as entropy, mutual information, and compression-based methods are commonly used to measure the complexity or uncertainty of data points. Entropy, for example, measures the level of unpredictability or disorder in a dataset, while mutual information quantifies the amount of shared information between different variables. Anomalous points are typically those that exhibit either unusually high or low information content compared to the rest of the data.

The process begins by calculating information-theoretic metrics for each data point. These metrics reflect how much information or complexity each point holds within the context of the dataset. Points with high entropy, indicating a high degree of disorder, or points that significantly deviate from typical mutual information values, are flagged as potential anomalies. Compression techniques can also be applied, as anomalous data is often harder to compress due to its distinct or unpredictable nature. Ultimately, this method is effective in fields like network intrusion detection, where unusual data patterns signal potential security threats, or in biological data analysis, where atypical gene expression levels might indicate anomalies in biological processes.

### 2.3.7. Spectral Based (8)

Spectral-based anomaly detection involves transforming data into a spectral domain to analyze its underlying structure using mathematical techniques such as eigenvalue decomposition. This method is particularly effective in identifying hidden patterns and detecting anomalies by examining the spectral properties of the data. By applying transformations such as Principal Component Analysis (PCA) or Singular Value Decomposition (SVD), spectral features like eigenvalues and eigenvectors can be extracted to represent the most dominant patterns in the dataset. Anomalies are identified based on deviations in these spectral features.

In the spectral domain, data points are projected onto a set of axes defined by the eigenvalues and eigenvectors. The key idea is that anomalies disrupt these dominant patterns, making them distinguishable when analyzed spectrally. For instance, outliers may appear as extreme values in the eigenvalue spectrum, or as deviations in the principal components that capture the majority of the variance in the data. The transformation into the spectral domain allows for a compact representation of complex data, which can be highly effective in identifying subtle anomalies that are not easily detectable in the original space. This approach is widely applied in domains such as network traffic monitoring, image data analysis, and sensor networks, where spectral properties can reveal deviations in normal patterns indicative of anomalies. (8)

### 2.3.8. Ensemble Based

Ensemble machine learning methods have emerged as powerful techniques for improving the predictive performance and robustness of models across various domains and tasks. In traditional machine learning, a single model is trained on a dataset to make predictions. However, ensemble methods take a different approach by combining the predictions of multiple base models to produce a final prediction, often achieving higher accuracy than any individual model alone. (3)

The underlying principle of ensemble methods is rooted in the concept of "wisdom of the crowd," where aggregating the opinions of multiple models can lead to more accurate and reliable predictions. By leveraging diversity among base models, ensemble methods can effectively capture different aspects of the data and reduce the risk of overfitting, leading to improved generalization performance.

Ensemble methods can be broadly categorized into two main types: bagging and boosting. Bagging methods, such as Random Forests, train multiple base models independently on different subsets of the training data and combine their predictions through averaging or voting. Boosting methods, such as AdaBoost and Gradient Boosting

Machines (GBM), sequentially train base models, with each subsequent model focusing on correcting the errors of its predecessors.

In addition to bagging and boosting, other ensemble techniques include stacking, where the predictions of base models serve as input features for a meta-model, and ensemble pruning, which aims to select a subset of base models for aggregation to improve computational efficiency.

Ensemble methods have demonstrated remarkable success across various machine learning tasks, including classification, regression, and anomaly detection. In 2021, an ensemble method called SAND was proposed, that could detect anomalies in timeseries datasets.

- **The Subsequence Anomaly Detection (SAND) Method**

To leverage the advantages of the previous technique, the method SAND was developed, suitable for subsequence anomaly detection in data streams. SAND builds a dataset of subsequences representing the different behaviors of the data series. These subsequences are weighted using statistical characteristics such as their cardinality (i.e., how many times the subsequence occurred) and their temporality (i.e., the time difference this subsequence has been detected for the last time). SAND enables this data structure to be updated from one batch to another, while being able to compute an anomaly score at every timestamp. Thus, SAND proposes a solution to the subsequences anomaly detection task on streaming data. SAND benefits from $k$-Shape , a state of the art data-series clustering method, which is extended to enable the clustering result to be updated without storing any of the previous subsequences. (19)

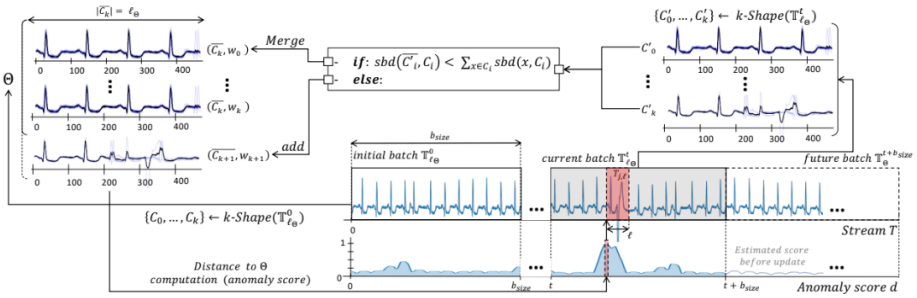


*Figure 14: A graphical representation of how SAND works (19)*

### 2.3.9. The Contestants

From the wide selection of machine learning algorithms, three have been selected to be used on my time series datasets. Each algorithm represents a different category of anomaly detection techniques, ensuring a comprehensive comparison. The chosen algorithms are:

- Isolation Forest (Classification based) (20)
- Local Outlier Factor (Nearest Neighbor based) (21)
- Subsequence Anomaly Detection (Ensemble) (19)

The selection of these algorithms ensures that we will have a representative from each of the most promising categories in anomaly detection. While the comparison results against a common dataset offer a general overview of the algorithms' efficiency, they cannot definitively classify their performance in all scenarios. Instead, they provide insight into how these methods might perform in similar contexts.

### 2.3.10. The Datasets

The datasets play a crucial role in testing, serving as the basis for evaluating the performance of the selected algorithms. In this project, we utilize multiple time series datasets to ensure a robust and comprehensive evaluation.

**Description of Datasets**

Our datasets consist of time series data collected from various sources. Each dataset includes many datapoints consisting of a value and a label (0 for normal, 1 for anomaly). The data spans a significant period, allowing us to capture various patterns and anomalies over time.

**Typical Preprocessing Steps** (22)

Data Cleaning: This involves handling missing values, removing duplicates, and filtering out irrelevant data points. Clean data is essential for accurate analysis. This step in our occasion is not needed.

Normalization: To ensure that the features are on a comparable scale, normalization is applied. This step is particularly important for algorithms like LOF, which rely on distance measures. This process takes place in the algorithm.

Segmentation: The time series data is segmented into smaller subsequences to facilitate analysis. This segmentation allows us to apply the Subsequence Anomaly Detection algorithm effectively. This process also takes place in the algorithm.

Labeling: For supervised evaluation, anomalies within the datasets are labeled. This labeling can be based on historical data, expert knowledge, or a combination of both. The labeling is achieved by using 1 and 0 flags .

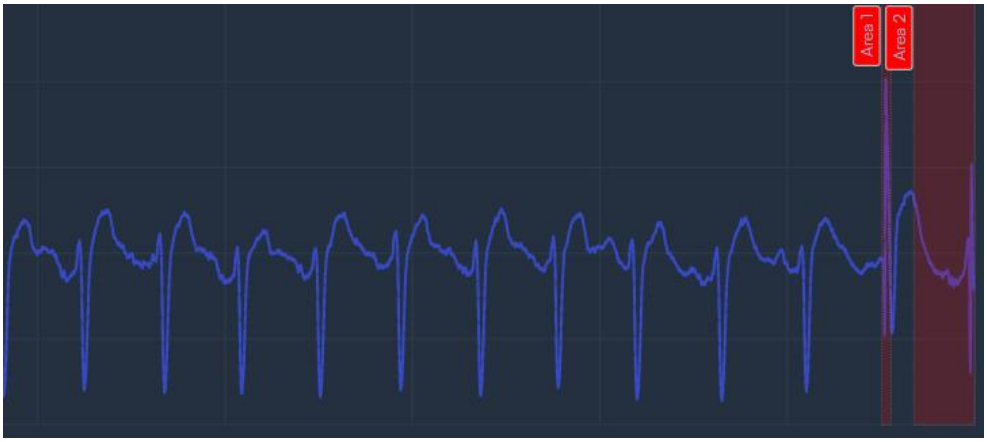


*Figure 15: A graphical representation of a timeseries dataset with anomalies*

**Types of Anomalies**

In time series data, anomalies can manifest in various forms. Identifying and categorizing these anomalies is crucial for a comprehensive evaluation of the algorithms used in this project. Below, we describe the common types of anomalies that are typically encountered in time series data:

Point Anomalies (Global Outliers): A point anomaly occurs when a single data point significantly deviates from the rest of the data. In time series datasets, this could be a sudden spike or drop in value at a particular timestamp that does not align with the expected pattern. For example, in a financial dataset, a sudden and large withdrawal or deposit might be flagged as a point anomaly. Algorithms like Isolation Forest are particularly effective at detecting such anomalies because they isolate these points in fewer splits compared to normal points.

Contextual Anomalies (Conditional Anomalies): Contextual anomalies occur when a data point is considered anomalous only in a specific context but normal in another. In time series data, context is typically defined by time or seasonality. For instance, an unusually high temperature might be anomalous in winter but normal during summer. Detecting contextual anomalies requires understanding the broader context in which the data point occurs, making it more complex than point anomaly detection. Time series models such as LSTM and algorithms considering seasonality are more suited for detecting contextual anomalies.

Collective Anomalies: Collective anomalies refer to a sequence or group of data points that together represent an anomaly, even though individual points within the sequence may appear normal (figure 15). These often arise in time series data when the relationship between consecutive points exhibits an abnormal trend or pattern over time. For instance, a sudden, consistent increase in network traffic over a few minutes might signal an attack, even if each individual point is within normal range. Detecting collective anomalies often requires algorithms capable of analyzing patterns across time, such as subsequence anomaly detection methods.

Transient Anomalies (Short-Lived Anomalies): Transient anomalies are anomalies that last for a short period and then return to normal. In time series datasets, these could be caused by temporary system malfunctions or brief environmental changes. Detecting transient anomalies is challenging because the anomaly could blend in with the normal pattern when considering long-term trends.

### 2.3.11. Algorithm Evaluation

Basic Evaluation Metrics: The metrics are based on these 4 datapoint characterizations after the evaluation of the algorithm.

True Positive (TP): The datapoint is correctly found as an anomaly

True Negative (TN): The datapoint is correctly found as normality

False Positive (FP): The datapoint is falsely found as an anomaly

False Negative (FN): the datapoint is falsely found as a normality

To compare the performance of the algorithms, several key metrics are used:

- Recall or TPR (True Positive Rate):
  The ratio of true positive anomalies to the total number of actual anomalies. High recall means the algorithm successfully identifies most of the anomalies present in the dataset.

$$TPR = TP/(TP + FN)$$

*Equation 12: The Recall Equation*

- FPR (False Positive Rate)
  The False Positive Rate measures the proportion of negatives incorrectly identified as positives by the classifier. It is calculated as:

$$FPR = FP/(FP + TN)$$

*Equation 13: The FPR Equation*

- Receiver Operating Characteristic (ROC) curve

  ROC analysis is a graphical approach for analyzing the performance of an algorithm. This methodology plots FPR on the x-axis and TPR on the y-axis for various values of anomaly threshold that a datapoint score must. The resulting plot can be used to compare the relative performance of different algorithms and to determine whether an algorithm performs better than random guessing.

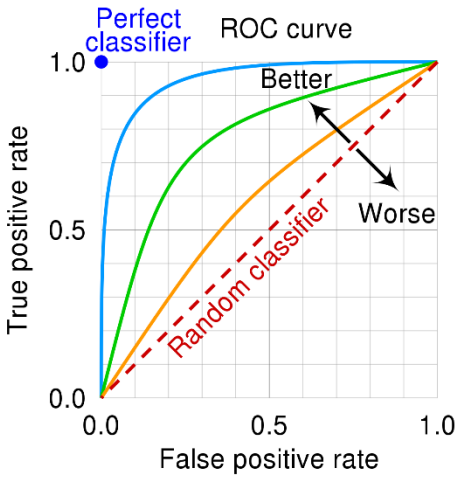


*Figure 16: Comparing 4 algorithms (classifiers) against a random one (23)*

- AUC (Area Under the Curve)
  The AUC measures the area under the ROC curve. It quantifies the overall ability of the model to discriminate between positive and negative classes. A higher AUC indicates better performance.

- Precision (P):
  Precision is the ratio of true positive anomalies to the total number of anomalies detected. High precision indicates that the algorithm accurately identifies anomalies without many false positives.

$$P = TP/(TP + FP)$$

*Equation 14: The Precision Equation*

- F1 Score (F1):
  The harmonic mean of precision and recall, providing a balanced measure of an algorithm's performance.

$$F1 = 2 \cdot (P \cdot TPR)/(P + TPR)$$

*Equation 15: The F1 Score Equation*

- AP (Average Precision)
  Average Precision summarizes the precision-recall curve and gives a single value that combines precision and recall across different thresholds. It is a more comprehensive measure than just precision or recall.

- Processing Time:
  The time taken by the algorithm to process the dataset. This metric is crucial for applications requiring real-time anomaly detection. (24)

### 2.3.12. Results

The results of our analysis are based on the performance of the three selected algorithms Isolation Forest, Local Outlier Factor, and Subsequence Anomaly Detection across multiple metrics. Each algorithm was tested on our time series datasets and produced a txt file that were populated with logs like that:

fileName:001_UCR_Anomaly_DISTORTED, AUC:0.38, Precision:0.03, Recall:0.08, F1:0.05, AP:0.01, tn_count:77712, fn_count:571, fp_count:1462, tp_count:50, elapsed_time:18.58 seconds, datapoint_count:79795

The results were evaluated using the performance metrics: AUC, Precision, Recall, F1 Score, Average Precision, and Elapsed Time. For our case we will demonstrate the most simple ones as follows:

| | Was anomaly | Was not anomaly | Alorithm |
|---|---|---|---|
| Predicted anomaly | 6013 | 182355 | Isolation Forest |
| Predicted anomaly | 14922 | 188260 | Local Outlier Factor |
| Predicted anomaly | 12423 | 116462 | Subsequence Anomaly Detection |
| Did not predict anomaly | 39345 | 14198233 | Isolation Forest |
| Did not predict anomaly | 30436 | 14192328 | Local Outlier Factor |
| Did not predict anomaly | 32935 | 14264126 | Subsequence Anomaly Detection |

*Table 1: The results of the algorithms upon the same 234 datasets*

Table 1 presents the performance of three anomaly detection algorithms—Isolation Forest, Local Outlier Factor, and Subsequence Anomaly Detection—evaluated across 234 datasets. The table shows the number of instances where each algorithm predicted an anomaly correctly (true positives), incorrectly predicted an anomaly when none existed (false positives), failed to predict an actual anomaly (false negatives), and correctly identified non-anomalies (true negatives).

Isolation Forest predicted 6,013 actual anomalies but incorrectly identified 182,355 normal instances as anomalies, yielding a very low precision of 0.032. This suggests that the model produces a high number of false positives, leading to inefficiencies in identifying relevant anomalies. Despite its low precision, the recall value of 0.13 shows that the algorithm has a reasonable ability to capture actual anomalies, although this still leaves many anomalies undetected (39,345 instances). The F1 score of 0.05 reinforces the observation that Isolation Forest is not particularly effective in this dataset.

Local Outlier Factor demonstrates a noticeable improvement over Isolation Forest, with 14,922 true positives and a relatively smaller number of false positives (188,260). Its precision, calculated at 0.073, while still low, is more than double that of Isolation Forest. More importantly, the recall of 0.33 indicates that the algorithm captures a much higher proportion of actual anomalies. With a false negative count of 30,436, it misses fewer anomalies than Isolation Forest. The F1 score of 0.12 suggests a more balanced performance but still highlights that the algorithm struggles with precision.

Subsequence Anomaly Detection shows the best balance between precision and recall, predicting 12,423 true anomalies while incorrectly flagging 116,462 normal instances. Its precision of 0.096 is the highest among the three algorithms, demonstrating a better ability to minimize false positives. The recall of 0.27, though lower than that of Local Outlier Factor, indicates that it still captures a substantial number of anomalies. The F1 score of 0.14, the highest among the three, suggests that Subsequence Anomaly Detection offers a more reliable trade-off between precision and recall in these datasets.

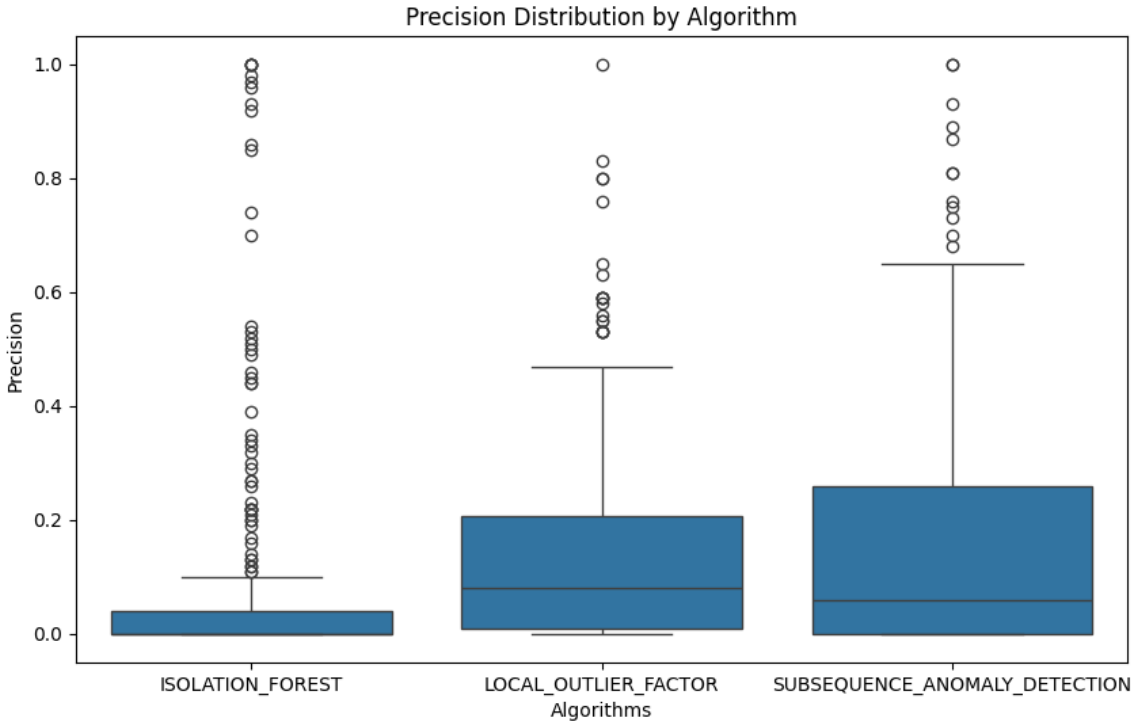


*Figure 17: Precision Distribution*

The precision distribution across the three algorithms is further visualized in the boxplot, which offers additional insights into the consistency and reliability of the models' performance across datasets.

**Isolation Forest** shows a tight clustering of precision values near the lower end, with very few outliers reaching higher precision levels. This indicates that the algorithm consistently struggles with precision across different datasets, further confirming its tendency to generate a high rate of false positives. The interquartile range is very small, highlighting that the variability in precision is minimal, but unfortunately skewed towards poor performance.

**Local Outlier Factor** presents a broader interquartile range in the boxplot, suggesting more variability in its precision across datasets. While the median precision remains relatively low, the wider spread shows that the algorithm performs better on some datasets compared to others. This variability points to a potential for tuning the model or selecting it for specific types of datasets where its performance could be optimized. The presence of numerous outliers, some with relatively high precision, indicates that while the overall performance is still suboptimal, there are instances where the model successfully minimizes false positives.

**Subsequence Anomaly Detection** displays the widest interquartile range and the highest median precision, signaling that this algorithm consistently performs better than the others in terms of precision. The distribution also includes several outliers reaching

significantly higher precision, suggesting that in certain datasets, the algorithm is capable of very strong performance. The broader spread of the boxplot compared to the other algorithms, however, indicates that the precision can vary substantially between datasets, meaning it is not uniformly reliable. Nonetheless, it offers the most promising balance between minimizing false positives and correctly identifying anomalies.

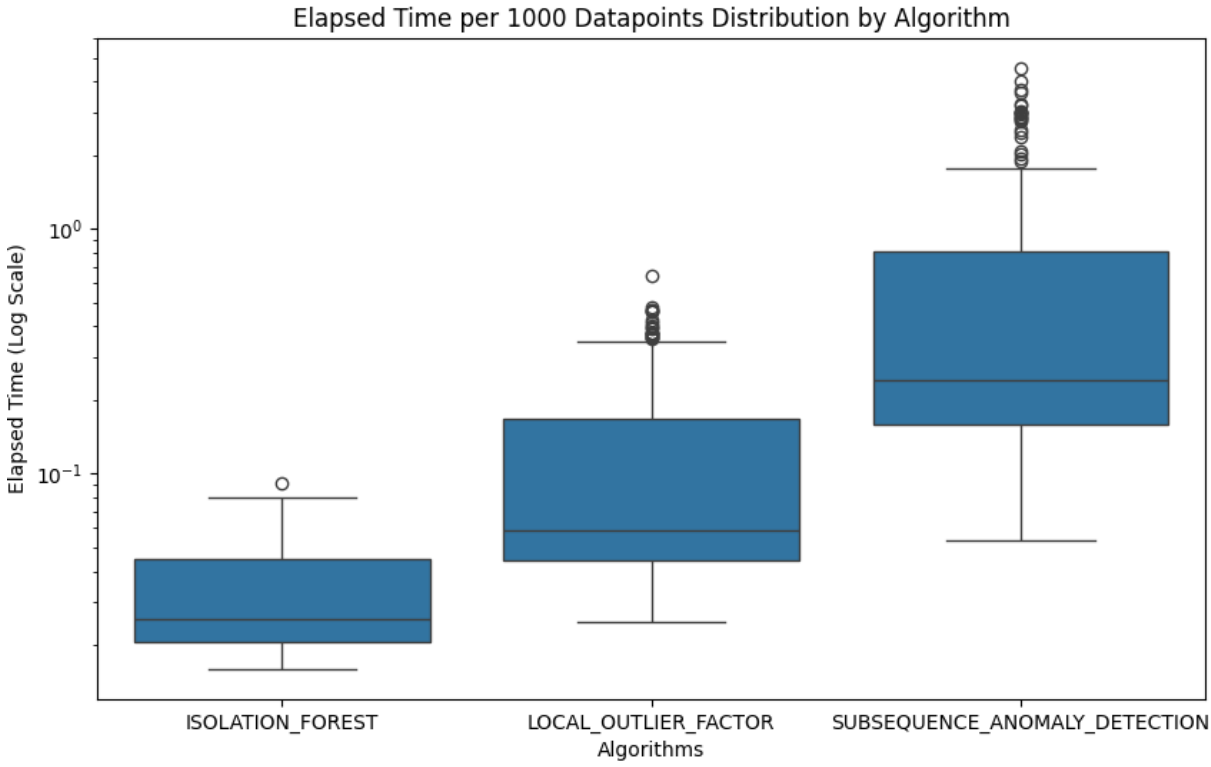


*Figure 18: Elapsed Time Distribution*

Figure 18 illustrates the distribution of elapsed time (in log scale seconds) for the three algorithms. The log scale of the y-axis emphasizes the differences in execution time, even for small variations.

**Isolation Forest** algorithm shows a relatively tight distribution of elapsed time, with most runs taking between 1 and 10 units of time (log scale). The median elapsed time is positioned around 1 unit, suggesting that the algorithm is the fastest of the three across the datasets. The absence of significant outliers implies consistent performance across datasets, with minimal variability in execution time.

**Local Outlier Factor** algorithm demonstrates a broader distribution of execution times compared to Isolation Forest. The median execution time lies around 10 units, indicating that LOF tends to take significantly longer than Isolation Forest on average. However, the presence of several outliers above 100 units shows that LOF can sometimes be inefficient or perform poorly on certain datasets. The wider interquartile range (IQR)

indicates a higher variability in its performance, with times ranging from 1 unit to over 100.

**Subsequence Anomaly Detection** algorithm has the widest distribution of elapsed times, with a median close to 10 units, similar to LOF. The larger spread in the boxplot, particularly the longer upper whisker, suggests that this algorithm often takes longer to run on some datasets, with several runs extending up to 100 units or beyond. Like LOF, it has multiple outliers, indicating that its execution time is less predictable and more dependent on the dataset characteristics.

The results provide a comprehensive comparison of the selected algorithms, highlighting their strengths and weaknesses across different metrics:

**Isolation Forest** is quick and computationally efficient but struggles with precision, leading to lower overall performance metrics. It may be suitable for applications where speed is critical and some trade-off in accuracy is acceptable.

**Local Outlier Factor** offers a balanced approach with moderate performance in both precision and recall. It shows variability across datasets but generally performs well, making it a versatile choice for various applications.

**Subsequence Anomaly Detection** consistently performs the best across most metrics but at the cost of longer processing times. Its robust detection capabilities make it ideal for scenarios where accuracy is paramount and computational resources are sufficient.

Based on the analysis, **Subsequence Anomaly Detection** emerged as the most effective algorithm for anomaly detection in my time series datasets. It provided the best balance of precision, recall, and overall performance metrics, despite its higher computational demands. Given enough computational power this is the preferred choice for deployment in my fleet monitoring system. The other algorithms also have their merits and could be considered depending on specific application requirements and constraints.

## 2.4. Applications

The aforementioned Machine Learning techniques offer us a more concise way to extract hidden information from large datasets and can be implemented in various fields, for example:

- **Social Media Features**

Social media platforms use machine learning algorithms and approaches to create some attractive and excellent features. For instance, Facebook notices and records your activities, chats, likes, and comments, and the time you spend on specific kinds of posts. Machine learning learns from your own experience and makes friends and page suggestions for your profile

- **Product Recommendations**

Product recommendation is one of the most popular and known applications of machine learning. Product recommendation is one of the stark features of almost every e-commerce website today, which is an advanced application of machine learning techniques. Using machine learning and AI, websites track your behavior based on your previous purchases, searching patterns, and cart history, and then make product recommendations.

- **Image Recognition**

Image recognition, which is an approach for cataloging and detecting a feature or an object in the digital image, is one of the most significant and notable machine learning and AI techniques. This technique is being adopted for further analysis, such as pattern recognition, face detection, and face recognition

- **Sentiment Analysis**

Sentiment analysis is one of the most necessary applications of machine learning. Sentiment analysis is a real-time machine learning application that determines the emotion or opinion of the speaker or the writer. For instance, if someone has written a review or email (or any form of a document), a sentiment analyzer will instantly find out the actual thought and tone of the text. This sentiment analysis application can be used to analyze a review based website, decision-making applications, etc.

- **Automating Employee Access Control**

Organizations are actively implementing machine learning algorithms to determine the level of access employees would need in various areas, depending on their job profiles. This is one of the coolest applications of machine learning.

- **Marine Wildlife Preservation**

Machine learning algorithms are used to develop behavior models for endangered cetaceans and other marine species, helping scientists regulate and monitor their populations.

- **Regulating Healthcare Efficiency and Medical Services**

Significant healthcare sectors are actively looking at using machine learning algorithms to manage better. They predict the waiting times of patients in the emergency waiting rooms across various departments of hospitals. The models use vital factors that help define the algorithm, details of staff at various times of day, records of patients, and complete logs of department chats and the layout of emergency rooms. Machine learning algorithms also come to play when detecting a disease, therapy planning, and prediction of the disease situation. This is one of the most necessary machine learning applications.

- **Predict Potential Heart Failure**

An algorithm designed to scan a doctor's free-form e-notes and identify patterns in a patient's cardiovascular history is making waves in medicine. Instead of a physician digging through multiple health records to arrive at a sound diagnosis, redundancy is now reduced with computers making an analysis based on available information.

- **Banking Domain**

Banks are now using the latest advanced technology machine learning has to offer to help prevent fraud and protect accounts from hackers. The algorithms determine what factors to consider to create a filter to keep harm at bay. Various sites that are unauthentic will be automatically filtered out and restricted from initiating transactions.

- **Language Translation**

One of the most common machine learning applications is language translation. Machine learning plays a significant role in the translation of one language to another. The technology behind the translation tool is called 'machine translation.' It has enabled people to interact with others from all around the world; without it, life would not be as easy as it is now. It has provided confidence to travelers and business associates to safely venture into foreign lands with the conviction that language will no longer be a barrier.

# Chapter 3

# 3. The Web Application

## 3.1. General overview

Building a web application involves several key components, each playing a crucial role in ensuring the application runs smoothly and efficiently. These components include the frontend, backend, database, cloud deployment, and version control. Understanding how these parts work together is essential for creating a successful web application.

- **Frontend: The User Interface (UI)**

The frontend is like the face of a watch – it's what users interact with directly. It includes everything you see and click on in a web application, such as buttons, forms, and images. The main technologies used for the frontend are HTML, CSS, and JavaScript, along with modern frameworks like React, Angular, and Vue.js.

User Experience (UX): The frontend aims to provide a smooth and easy-to-use experience. This means designing a clear layout, making sure the application loads quickly, and ensuring it responds to user actions in real-time.

Responsive Design: With users accessing web applications on various devices, from phones to desktops, it's important that the application looks good and works well on all screen sizes.

Performance Optimization: Techniques like lazy loading (loading content only when needed) and code splitting (dividing the code into smaller parts) help make the frontend faster and more efficient.

- **Backend: The Server-Side Logic**

The backend is like the internal mechanism of a watch – it powers the application behind the scenes. It processes data, handles business logic, and manages server-side operations. The backend ensures that data is processed correctly and sent to the frontend when needed.

Server-Side Logic: This includes processing user inputs, managing user authentication (login and security), and handling requests between the client (user's browser) and server.

APIs (Application Programming Interfaces): APIs allow different parts of the application to communicate and share data. They make it easier to develop and manage the backend by breaking it into smaller, reusable pieces.

Performance and Scalability**:** The backend must be able to handle many users at once and scale up as needed to maintain performance.

- **Database: The Data Storage**

The database is like the memory of a watch – it stores all the data that the application needs to function. Choosing the right database and organizing it effectively is crucial for the application's efficiency.

Data Storage and Retrieval: Databases store user information, application data, and other important information, making it available when needed.

Types of Databases: Depending on the application, different databases can be used, such as relational databases (e.g., PostgreSQL, MySQL) or NoSQL databases (e.g., MongoDB, Firebase).

Real-Time Data: For applications needing instant data updates, databases with real-time capabilities (e.g., Firebase, Supabase) are used to ensure data is always up-to-date.

- **Cloud Deployment: Hosting and Scaling**

Cloud deployment is like the watchmaker who assembles, maintains, and ensures the watch runs smoothly. It involves hosting the web application on cloud servers, providing the infrastructure needed to run, scale, and manage the application efficiently.

Scalability and Flexibility: Cloud platforms like Vercel, AWS, and Google Cloud allow the application to handle varying loads and grow as needed.

Continuous Deployment (CI/CD): With cloud deployment, automated systems can be set up to deploy updates and changes seamlessly.

Cost Efficiency: Cloud services often offer cost-effective solutions with pay-as-you-go models, reducing the need for significant upfront investment in infrastructure.

- **Version Control: Managing Code Changes**

Version control is like the record-keeper for your watch assembly process – it tracks all changes made to the application's code over time. This is essential for collaboration, backup, and recovery.

Git: The most widely used version control system, Git allows developers to track changes, collaborate on code, and revert to previous versions if needed.

Repositories: Platforms like GitHub, GitLab, and Bitbucket host code repositories, making it easy to share code, manage contributions from multiple developers, and maintain a history of changes.

Branching and Merging: Developers can create branches to work on new features or fixes without affecting the main codebase. Once the changes are ready, they can be merged back into the main branch.

- **Integration of Components**

Just as the precise interaction of gears, springs, and other parts is necessary for a watch to keep accurate time, the integration of the frontend, backend, database, cloud deployment, and version control is essential for a web application to function smoothly. Each part must communicate effectively to ensure data flows seamlessly and the application remains responsive and reliable.

Data Flow: Ensuring data moves smoothly between the frontend, backend, and database is crucial. APIs and data-fetching mechanisms play a key role in this process.

Synchronization: Real-time synchronization between components ensures that all parts of the application are up-to-date, providing a consistent user experience.

Monitoring and Maintenance: Continuous monitoring and maintenance of the cloud infrastructure and application components ensure optimal performance and quick issue resolution.

## 3.2. Advanced Architectural Models, different topics for consideration and Emerging Trends in Web Application Development

### 3.2.1. Advanced Architectural Models

- **Cloud Service Models**

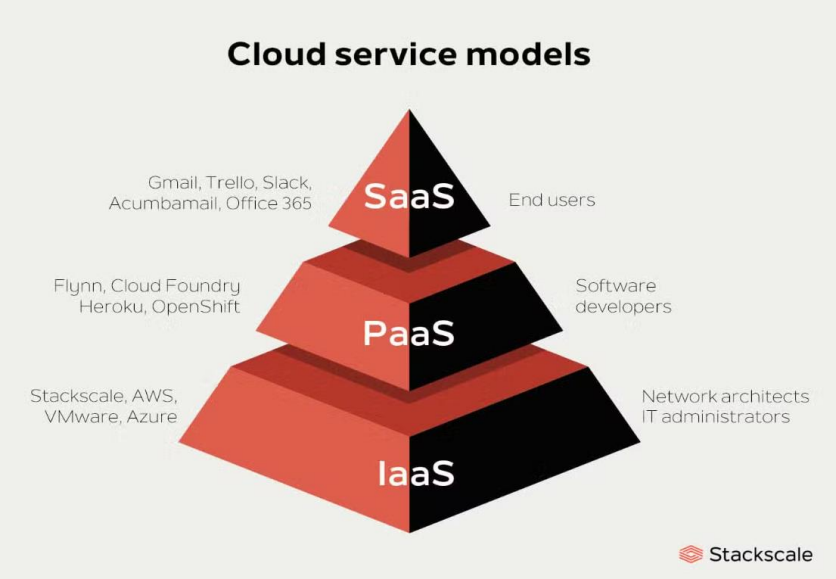


*Figure 19: Levels of Architectural Models (25)*

Cloud service models have revolutionized how web applications are developed, deployed, and maintained. Understanding the differences between Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS) is crucial for selecting the right solution for specific application needs. Each model offers unique benefits and challenges, influencing the scalability, cost, and management of web applications.

- Software as a Service (SaaS)

SaaS provides software applications over the internet on a subscription basis. Users can access these applications via a web browser, without worrying about the underlying infrastructure. Examples of SaaS include Google Workspace, Microsoft 365, and Salesforce. The primary advantages of SaaS are reduced time to benefit, lower costs, scalability, and accessibility from any location. However, SaaS also has

disadvantages such as limited control over the software, potential security concerns, and dependency on internet connectivity.

- Infrastructure as a Service (IaaS)

IaaS offers virtualized computing resources over the internet. It provides the basic building blocks for cloud IT and typically includes servers, storage, and networking. Examples of IaaS providers are Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. The main advantages of IaaS are high scalability, cost efficiency, flexibility, and full control over the infrastructure. On the downside, IaaS requires in-depth technical knowledge to manage the infrastructure and poses potential security risks.

- Platform as a Service (PaaS)

PaaS provides a platform allowing customers to develop, run, and manage applications without dealing with the underlying infrastructure. Examples include Heroku, Google App Engine, and Microsoft Azure App Service. The advantages of PaaS are simplified development, reduced management burden, and support for multiple programming languages. However, PaaS users may experience limited control over the environment and potential vendor lock-in.

- **Popular Architectures: Monolithic vs. Microservices**

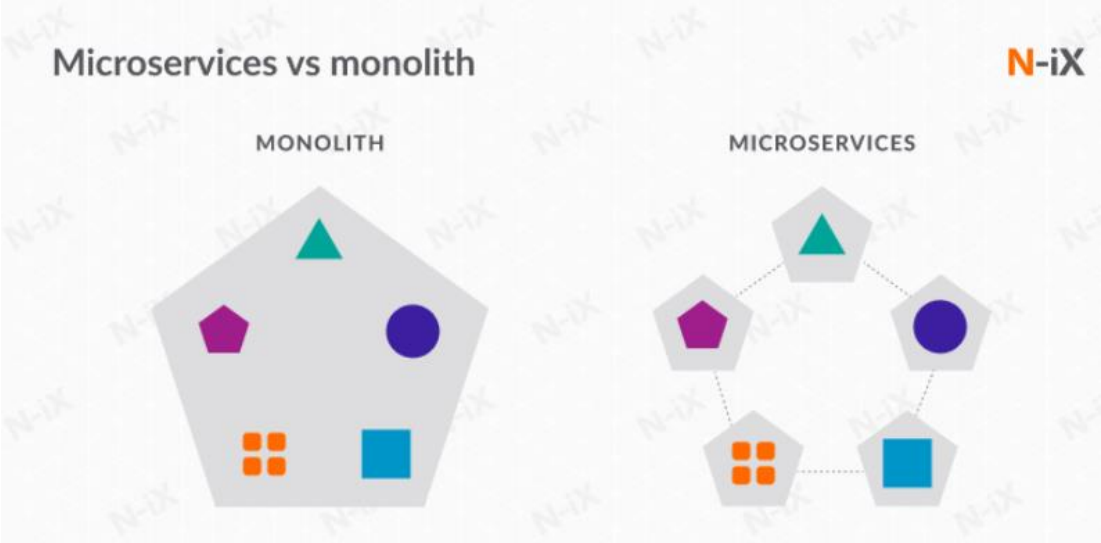


*Figure 20: Distribution of the percentage of each algorithm to the total AUC (26)*

The architecture of a web application significantly impacts its development, deployment, and scalability. Monolithic and microservices architectures represent two contrasting approaches. Monolithic architecture is a traditional, single-unit approach, while microservices architecture breaks down the application into smaller, independent services. Understanding these architectures helps in choosing the right approach based on application requirements, team structure, and long-term maintenance considerations.

- Monolithic Architecture

  Monolithic architecture is a traditional model of software development where all components of an application are bundled together into a single package. The advantages of this approach include simplicity in development and deployment, as well as easier debugging and testing. However, monolithic architecture also has disadvantages, such as limited scalability, difficulty in maintenance and updates, and the potential for a single point of failure.

- Microservices Architecture

  Microservices architecture structures an application as a collection of loosely coupled services, each of which implements a specific business capability. The benefits of microservices include improved scalability, easier deployment and maintenance, and better fault isolation. However, this architecture also introduces increased complexity, a need for robust monitoring and management, and potential latency issues due to inter-service communication.

### 3.2.2. Topics for consideration

- **Security: Safeguarding the Application**

Security is a fundamental aspect of web application development, protecting both user data and the application itself from potential threats. Implementing robust security measures, such as data encryption, strong authentication, and regular security audits, is essential to prevent breaches and ensure user trust. Best practices in web application security include implementing SSL/TLS for data encryption, using strong authentication mechanisms, regularly updating dependencies, and performing security audits.

- **DevOps Practices: Enhancing Development and Operations**

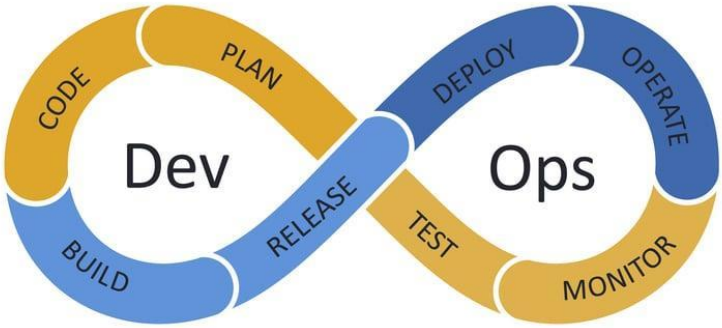


*Figure 21: : DevOps Steps (27)*

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the software development lifecycle while delivering high-quality software continuously. It emphasizes collaboration and communication between developers and operations teams, automation of processes, and continuous integration and continuous delivery (CI/CD). The goal of DevOps is to improve the speed, efficiency, and reliability of software deployment, ensuring that updates and new features can be released frequently and with minimal disruption. This approach enables organizations to respond quickly to market changes and customer needs, maintaining a competitive edge.

  - **Plan:** Develop a strategy, define objectives, create timelines, and allocate resources for the project.
  - **Code:** Write and review the source code using version control systems, collaborating with team members

- **Build:** Compile source code into executable formats using build automation tools, integrating contributions from different developers.
- **Test:** Conduct automated and manual testing to identify bugs, ensuring code quality and security while validating requirements.
- **Release:** Prepare software for deployment with final quality checks, deploying to production or staging environments.
- **Deploy:** Roll out software to live environments using deployment automation tools, ensuring minimal downtime and rollback capabilities.
- **Operate:** Maintain and manage the live software, monitoring performance, and ensuring smooth operation.
- **Monitor:** Continuously observe system performance, logging, and analyzing data to identify and resolve issues proactively.

### 3.2.3. The Future of Web Applications

The landscape of web application development is continually evolving, driven by emerging trends and technologies. Progressive Web Apps (PWAs) and serverless computing are two significant innovations shaping the future of web applications. PWAs combine the best of web and mobile apps, offering features like offline access, push notifications, and fast loading times, which enhance user experience, cross-platform compatibility, and performance. Serverless computing allows developers to build and run applications without managing the underlying infrastructure, with examples including AWS Lambda, Azure Functions, and Google Cloud Functions. The advantages of serverless computing include reduced operational complexity, cost efficiency, and automatic scaling. Incorporating these emerging trends and technologies can significantly enhance the functionality and user experience of modern web applications.

## 3.3. My Cool Dashboard

### 3.3.1. Front-end Back-end

The Web Application which was created for the needs of this thesis, is based on the Next.js framework. This framework is an evolution of the React.js Library, which uses JavaScript as main language. The selection of this framework is personal, as we want to discover and leverage the advantages Next.js has to offer. Some of them are:

- **Server-Side Rendering (SSR):**

  Enhances performance and SEO by generating pages on the server, ensuring faster load times and better indexing by search engines.

- **Static Site Generation (SSG):**

  Creates static HTML at build time, improving performance and scalability.

- **Hybrid Approach:**

  Combines SSR and SSG, allowing pages to be statically generated or server-rendered as needed.

- **Automatic Code Splitting:**

  Loads only the necessary code for each page, reducing load times.

- **Image optimization:**

  Automatically optimizes images on-demand. This includes resizing, format conversion, and lazy loading.

- **Server Actions:**

  Server Actions are asynchronous functions that are executed on the server, substituting on some degree the API calls

- **API Routes:**

  Integrates backend functionality within the same project, simplifying development.

- **File-Based Routing:**

  Easy to manage routes without additional configuration.

- **Developer Experience:**

  Hot-reloading, TypeScript support, and extensive plugin ecosystem streamline development.

- **Optimized for deployment on Vercel:**

  Seamless deployment and hosting integration with Vercel.

By using this framework, the Developer can work both on front and back end in the same time. This approach results in a more monolithic architecture compared to a microservices-based one.

### 3.3.2. Database

The next crucial component is the Database. The choice of database is personal and subjective. We opted for Supabase due to its versatile and user-friendly integration with my development environment, facilitated by the Supabase library. Some of the advantages are:

- **Versatility and Ease of Use:**

  Supabase provides a straightforward API and real-time capabilities, making it a powerful choice for modern web applications.

- **Seamless Integration:**

  The Supabase library integrates effortlessly with various development environments, reducing setup complexity and enhancing productivity.

To further streamline database development, we utilize *ORM (Object Relational Mapping) technology*. ORM tools create a bridge between object-oriented programming and relational databases, making database interactions more seamless. One of the most widely used ORM tools is Prisma, which simplifies database management and enhances development efficiency. The main reasons to choose this technique are:

- **Simplified Database Interactions:**

  ORMs allow developers to interact with the database using their preferred programming language's syntax, abstracting complex SQL queries.

- **Consistency and Maintainability:**

  ORMs help maintain consistency in database operations and improve code maintainability by using a unified interface for database interactions.

- **Prisma:**

  Prisma stands out as a popular ORM tool, offering robust features like type-safe database access, an intuitive query language, and seamless integration with various databases.

### 3.3.3. Cloud Deployment

Deploying a web application is a critical step that ensures your app is accessible to users globally. Vercel is an exceptional platform for this purpose, especially for applications built with Next.js, because this Platform as a service (PaaS) ensures:

- **Seamless Integration with Next.js:**

Vercel is designed to work effortlessly with Next.js, offering built-in support for server-side rendering (SSR) and static site generation (SSG).

Automatic optimizations for Next.js applications enhance performance and scalability without additional configuration.

- **Scalability and Performance:**

Vercel's global content delivery network (CDN) ensures fast load times by caching static assets at the edge, close to users.

The platform automatically scales your application to handle increased traffic, maintaining performance during high-demand periods.

- **Continuous Deployment (CI/CD):**

Integration with GitHub, GitLab, and Bitbucket allows for continuous integration and deployment. Changes pushed to the repository are automatically deployed, ensuring your application is always up-to-date.

Vercel’s preview URLs feature enables you to review changes in a live environment before merging them into the production branch.

- **Serverless Functions:**

Vercel supports serverless functions, allowing you to deploy backend logic without managing servers. This includes API routes and other server-side functionalities, enabling a full-stack development experience within a single platform.

- **Developer Experience:**

Vercel’s CLI and dashboard provide intuitive tools for managing deployments, environment variables, and project settings.

Automatic HTTPS, custom domains, and advanced security features ensure your application is secure and accessible.

- **Optimizations and Analytics:**

Built-in performance analytics offer insights into your application's performance, helping identify and address potential issues.

Automatic image optimization and lazy loading improve load times and user experience.

### 3.3.4. Version Control: GIT as Remote Repository

Version control is a fundamental aspect of modern software development, and Git is one of the most popular systems for this purpose. Utilizing Git as a remote repository offers several advantages, particularly when integrated with platforms like GitHub, GitLab, or Bitbucket. The Advantages of Git as a Remote Repository are:

- **Distributed Version Control:**

Git allows every developer to have a complete copy of the project history on their local machine. This distributed nature means you can work offline and commit changes locally, syncing with the remote repository when you're back online.

- **Collaboration:**

Git supports collaborative workflows, enabling multiple developers to work on the same project simultaneously. Features like branching, merging, and pull requests streamline collaboration and code review processes.

- **Branching and Merging:**

Git's branching model allows developers to create isolated environments for new features, bug fixes, or experiments. Branches can be merged back into the main codebase once the work is completed and reviewed, ensuring that the main branch remains stable.

- **History and Version Tracking:**

Git maintains a detailed history of all changes made to the codebase. Each commit records who made the change, when it was made, and what was changed, providing a comprehensive audit trail.

- **Integration with CI/CD Pipelines:**

Git integrates seamlessly with Continuous Integration and Continuous Deployment (CI/CD) pipelines. Changes pushed to the repository can trigger automated builds, tests, and deployments, ensuring that code is continuously tested and delivered.

- **Issue Tracking and Project Management:**

Platforms like GitHub, GitLab, and Bitbucket offer integrated issue tracking, project boards, and other management tools. This integration helps teams track progress, manage tasks, and address issues within the same ecosystem.

- **Security and Permissions:**

Git repositories hosted on platforms like GitHub provide robust security features, including branch protection, code review requirements, and fine-grained access controls. These features help safeguard the codebase and ensure that only authorized changes are made.

The main actions we can use to achieve version control are:

- **Clone**: Copy a remote repository to your local machine.
- **Commit**: Save changes to your local repository with a message.
- **Push**: Upload local commits to a remote repository.
- **Pull**: Fetch and merge changes from a remote repository to your local branch.
- **Branch**: Create a new branch to work on a feature.
- **Merge**: Combine changes from one branch into another.
- **Status**: Check the current state of your repository.
- **Add**: Stage changes for the next commit.
- **Log**: View the commit history.
- **Revert**: Undo a specific commit.

### 3.3.5. Messaging System

Until now we have presented the main components that comprise the Web Application. The system, except from the central hub that will store and show the anomaly events, has to be able to communicate with the sensors that will be planted in remote places. This could be implemented via APIs and http requests, but a better method exists that is more suitable for our use case

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. It operates on a publish/subscribe model, where devices (clients) can publish messages to specific topics and subscribe to topics to receive messages. This decouples the message producers from the consumers, enhancing flexibility and scalability.

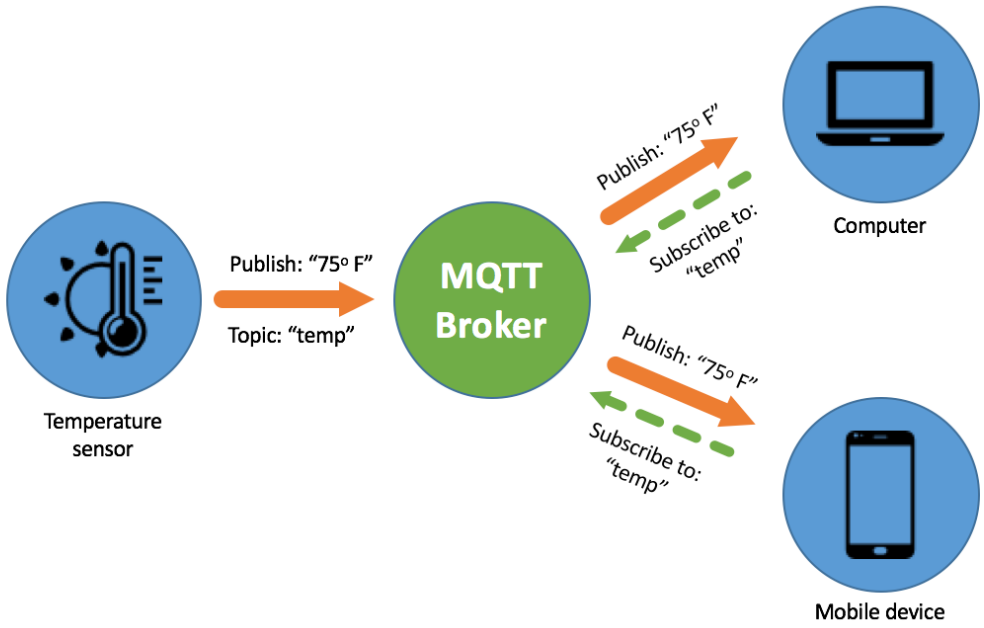


*Figure 22: Messaging system overview (28)*

Choosing MQTT over traditional APIs can be advantageous in certain scenarios, particularly in IoT and real-time communication applications. Here are some reasons why you might choose MQTT:

- **Low Bandwidth Usage:**

  MQTT is lightweight and minimizes network bandwidth usage, making it ideal for environments with limited resources.

- **Real-Time Communication:**

  It supports real-time communication with low latency, ensuring quick delivery of messages.

- **Scalability:**

  Efficiently handles many devices and high-frequency messaging, scaling better in IoT environments.

- **Reliability:**

  Offers various levels of Quality of Service (QoS), ensuring message delivery reliability.

- **Efficient Battery Use:**

  Suitable for battery-powered devices due to its low power consumption.

- **Publish/Subscribe Model**:

  Decouples senders and receivers, simplifying the architecture for complex messaging patterns.

In my case, the temperature sensor of the image will be my Anomaly Report Fleet (raspberries), the MQTT Broker of my choice will have to be configured, and the messages receiver will be my Web Application.

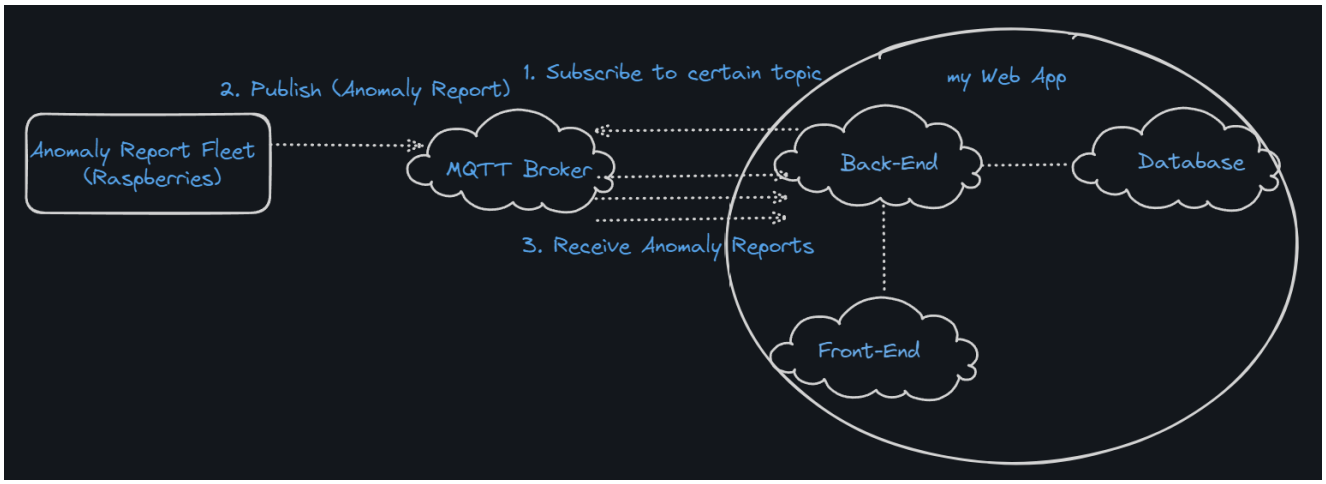


*Figure 23: Messaging system integration with my Web App*

### 3.3.6. How the App works

The main point of my thesis is to build a Web App that will display the anomalous data found by my sensor fleet, while being as simple as possible to use. The App is hosted on this Link and its purpose is only to give a visual representation of all the anomalies that have been found and reported to the backend, and sequentially to the database. Firstly, we are greeted from the signup/login page as figure 24 shows:

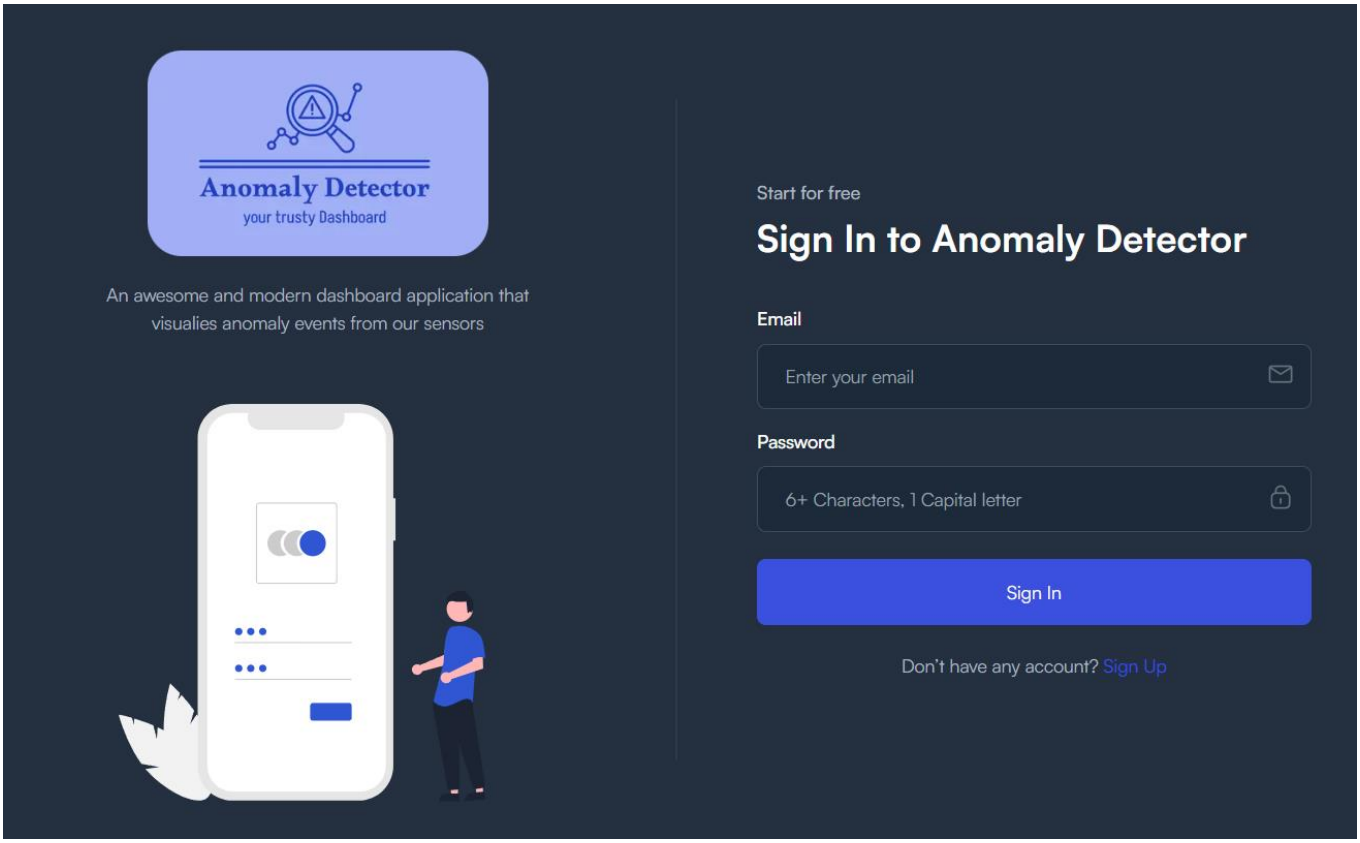


*Figure 24: Sign In page*

After a successful signin (or signup if this is the first time using the App), we will be redirected to the Dashboard as figure 24 shows:

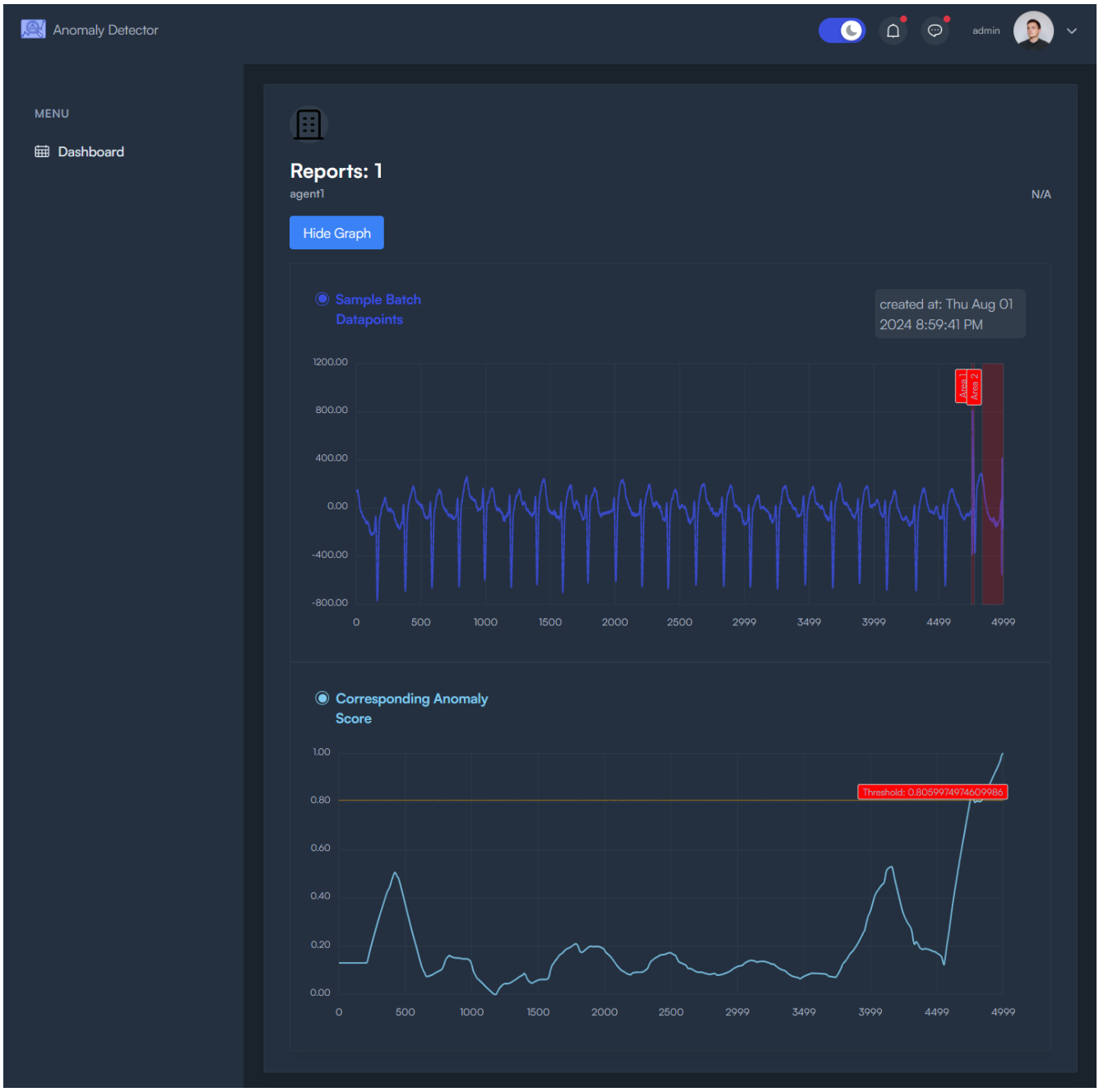


*Figure 25: Web App dashboard*

In the Web App Dashboard, we see that we have 1 report from agent 1 which is the name of one edge device from the fleet. In the top diagram we see 2 areas of the batch that were characterized as anomalies, and on the second diagram we see the

corresponding anomaly scores. The orange horizontal line is the threshold that separates the normal and anomalous values.

# Chapter 4

# 4. Deployment at Scale

## 4.1. What is Deployment at Scale

Deployment at scale means putting software applications or services into a large and flexible infrastructure that can handle heavy workloads and support rapid growth. Unlike traditional methods suitable for smaller projects, deploying at scale requires careful management of resources. This involves using containerization for consistent application delivery, load balancing to distribute traffic and avoid bottlenecks, and auto-scaling to adjust resources based on demand. A microservices architecture can improve both scalability and maintainability, while effective monitoring and logging help identify performance issues. Continuous integration and deployment (CI/CD) pipelines streamline updates and version control. While many organizations choose Kubernetes for resource management, this thesis suggests using Portainer to effectively orchestrate resources and achieve the necessary scalability for complex systems. By scale we refer to the scalability of number of devices we use, rather than the resource percentage of each device itself

In the landscape of modern software development, where agility, resilience, and scalability are paramount, deployment at scale holds immense significance. Several factors contribute to its importance:

- User Expectations: In an era where users expect seamless experiences and uninterrupted service availability, deploying at scale ensures that applications can handle increased user loads without compromising performance or reliability.
- Business Agility: Rapid deployment and scalability are critical for businesses to stay competitive. With deployment at scale, organizations can quickly respond to changing market demands, roll out new features, and adapt to evolving customer needs without lengthy downtimes or disruptions.
- Cost Efficiency: Efficient resource utilization is essential for optimizing costs, especially in cloud environments where resources are provisioned and billed based on usage. Deployment at scale allows for dynamic resource allocation, ensuring that resources are utilized optimally to minimize expenses.

- Resilience and Fault Tolerance: Large-scale deployments require robust architectures capable of withstanding failures and maintaining service continuity. By distributing workloads across redundant infrastructure and implementing failover mechanisms, deployment at scale enhances resilience and ensures high availability.
- Global Reach: With the rise of global markets and distributed teams, applications need to be deployed across geographically dispersed regions to provide low-latency access to users worldwide. Deployment at scale facilitates the deployment of distributed architectures that can serve users from diverse geographical locations.
- Support for Continuous Delivery: In the realm of continuous integration and continuous delivery (CI/CD), deploying at scale enables organizations to automate the deployment pipeline, from development to production, allowing for rapid and frequent releases while maintaining stability and reliability.

## 4.2. Using Deployment at Scale for my Project

Deploying machine learning software for anomaly detection on edge devices at scale requires a well-structured deployment strategy tailored to the specific challenges and constraints of edge computing environments. Implementing scalable deployment in my project allows for efficient management of a large fleet of edge devices while ensuring optimal performance, reliability, and scalability. By utilizing infrastructure orchestration tools and continuous integration and deployment pipelines, we can automate the deployment process, streamline updates, and maintain consistency across the distributed edge infrastructure. This approach facilitates rapid deployment of ML models to edge devices and enables seamless scaling to accommodate fluctuations in demand and the addition of new devices. Furthermore, incorporating security measures and monitoring capabilities into my deployment strategy ensures the integrity, privacy, and operational efficiency of the anomaly detection system deployed on edge devices. By effectively applying scalable deployment principles, my project can fully leverage the potential of edge computing for real-time anomaly detection and reporting, thereby providing organizations with actionable insights and enhanced operational efficiency.

For my project, we considered several alternatives that provide the capability to manage a fleet with large numbers of edge devices:

- **AWS IoT Greengrass**: AWS IoT Greengrass extends the capabilities of AWS to the edge, enabling devices to perform computing tasks locally. This allows devices to operate even without constant internet connectivity, which is crucial for IoT devices. However, the initial setup can be complex, and once implemented, the system is heavily integrated into the AWS ecosystem.

- **Portainer**: Portainer offers a user-friendly interface for managing Docker containers and clusters. It simplifies container management but may lack some advanced features required for large-scale deployments. Portainer is most suitable for projects that primarily use containers and require a straightforward management solution.
- **Balena**: Balena provides comprehensive tools for deploying and managing IoT applications across various hardware types. It is highly versatile and feature-rich but can be costly for large-scale deployments and ties users into the Balena ecosystem.
- **Azure IoT Edge**: Azure IoT Edge integrates seamlessly with Azure services, offering container-based deployment for edge applications. It is ideal for users already invested in the Azure ecosystem but may require additional effort to learn and implement.
- **AWS Fargate**: AWS Fargate manages container infrastructure, allowing developers to focus on application deployment without worrying about the underlying infrastructure. It offers automatic scaling, but users may experience higher costs and reduced control over the environment.
- **Ansible**: Ansible is an agentless tool that provides extensive capabilities for managing edge devices across diverse platforms. While powerful, it may require significant time to master, particularly for complex deployments. Ansible excels in deployment tasks but may need additional tools for monitoring and reporting.
- **Mender**: Mender specializes in over-the-air (OTA) updates, ensuring IoT devices remain up-to-date and secure. While effective for updates, it may need to be supplemented with other tools for comprehensive device management. Some advanced features may incur additional costs.

Each of these tools has distinct strengths and weaknesses, and the choice depends on specific needs and preferences. Considering the simplicity, community support, friendly UI, financial and time constraints of my project, we decided to use **Portainer**. This framework is suitable for the limited deployment scale of my thesis and offers a shallow learning curve, making it time-efficient and practical, by covering my requirements.

## 4.3. Portainer overview

Portainer is a user-friendly management platform designed to simplify the administration of Docker containers and clusters. It provides a graphical interface that abstracts the complexities of Docker's command-line interface, making container management accessible to users of all skill levels. With Portainer, users can efficiently deploy, monitor, and manage containerized applications across various environment platforms. Its intuitive interface allows users to perform tasks such as container creation,

image management, and resource monitoring with ease. Portainer's versatility and ease of use make it a valuable tool for developers, system administrators, and DevOps teams seeking to streamline their container orchestration workflows. It is important to say that for our case, the orchestration refers to the number of the devices we will use, and not to the resource allocation management of each device.

At a first glance this framework seems complicated, but in the following sections we explain more thoroughly with images. First, the overviews will be simple, but as we progress, we will add even more details. The simplest way to describe how it works is shown in the following image:

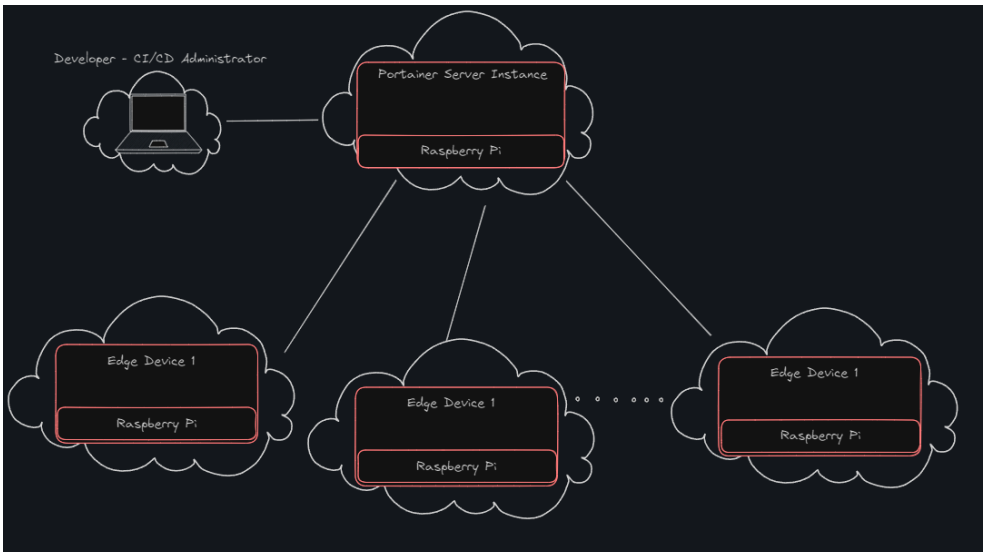


*Figure 26: Portainer general idea*

According to figure 26:

- **Portainer Server Instance:** This container will be the most crucial part of the system. Its responsibility will be to orchestrate the software deployment and update the remote edge devices, according to the Administrator's commands.

- **CI/CD Administrator:** The Administrator will be located remotely in relation to the server

- **Edge Device:** This is the device deployed in the field that will run the software as independent as possible. The will be no direct communication between the edge devices and the administrator

## 4.4. Portainer at my service

Now that you hopefully understood how this framework works, let's see how to harness its potential for the distributed part of my system. As mentioned in Chapter 3, the Portainer Server and Portainer Edge devices will be Raspberry pies (aarch64 architecture). These devices will not be reachable from WAN, so we must find a way to connect to the Server remotely. Thankfully, Portainer has provided us (29) a secure and efficient way to deploy for my use case. The prerequisite for this is to buy a domain name with which we will be able to reach the remote Portainer Server. This domain will also make it possible for the Portainer Edge fleet to communicate with the Server and update their code from an online repository as instructed by the administrator. Figure 26 gives a visual representation of the connecting nodes.

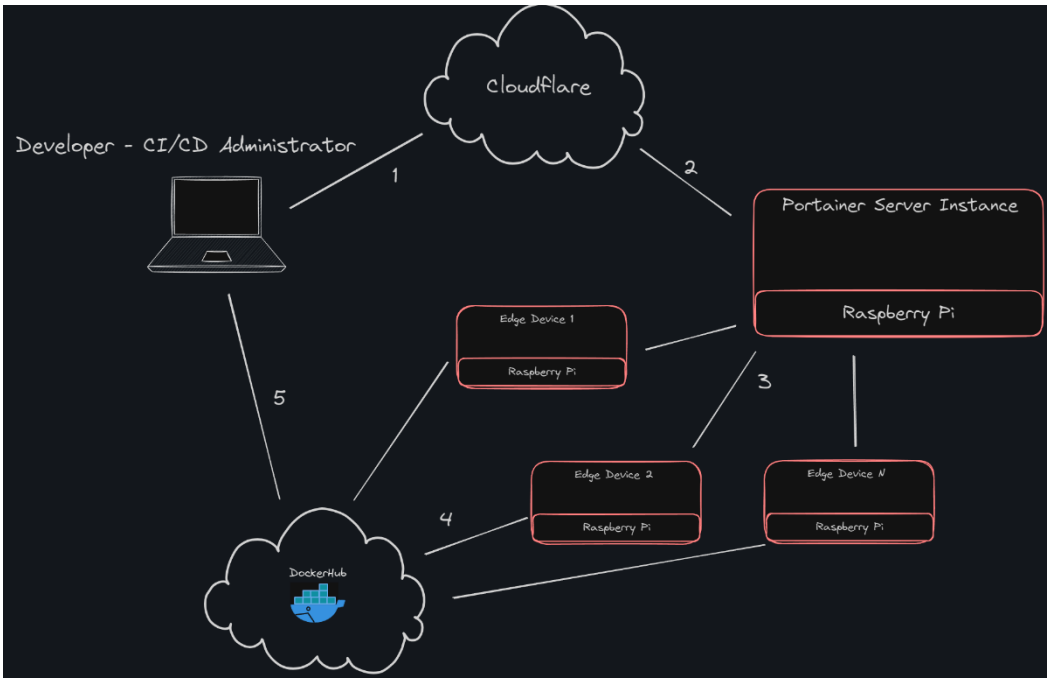


*Figure 27: My CI/CD pipeline*

1. This is where we use the prerequisite domain. At first, we create a tunnel by running the *docker run* command we get from Cloudflare, in the Portainer Server Instance. The --Token argument we pass is enough for the container to create and establish the tunnel connection.

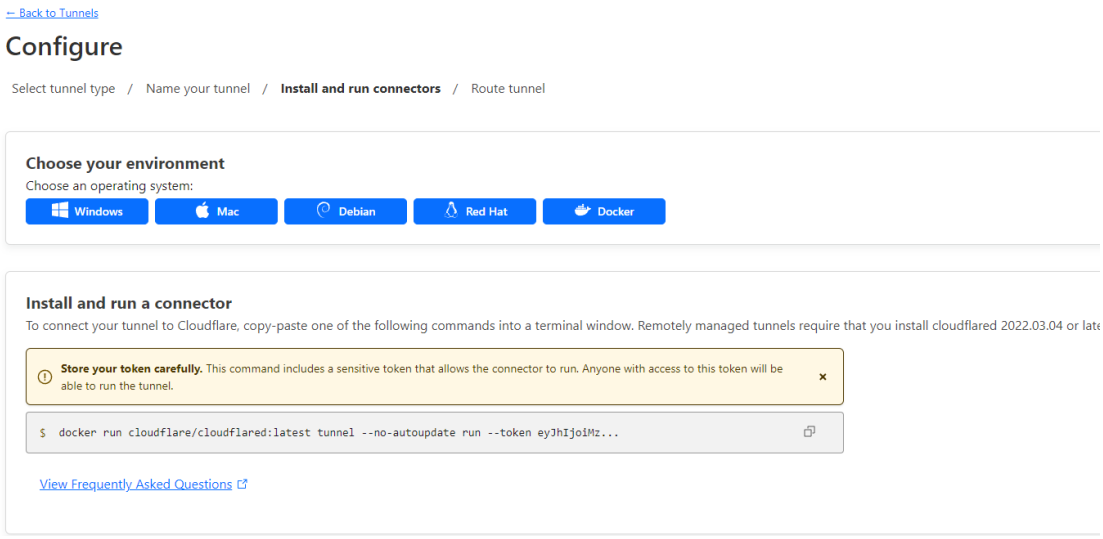


*Figure 28: Tunnel configuration to reach effectively the Portainer Server Instance*

In Figure 29 we can see the 2 configured tunnels we have created (Their status will be DOWN, unless we run them on the Portainer Server Instance)

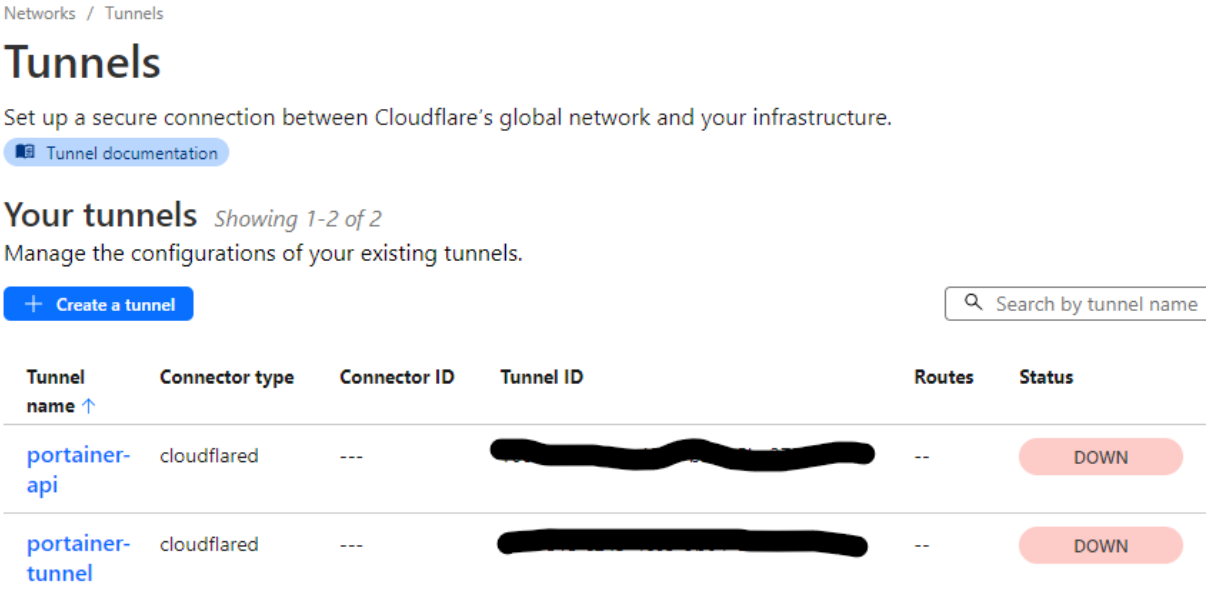


*Figure 29: The 2 active tunnels needed*

2. After a brief configuration on the Cloudflare service, when we visit my domain, a secure tunnel gets created to the running instance of the Portainer Server, giving me real time access to the Portainer UI, as if the server was locally deployed.
3. In this step we let edge devices (agents) join my fleet. Figure 30 in GIF format shows the UI of the Portainer Server. In the first frames you see we have access to the local (server) environment, as well as the environment of an Edge Device we have already

set up. To expand my fleet, we start by configuring the token and the container in general that will run in the remote Edge Device. The token has the addresses (the tunnel addresses from step 1) that are essential for the device to join our fleet. After a successful run of this container on an Edge Device, we will find this device in the waiting room (also shown in gif in the side panel), ready to be trusted. Once trusted, we will be able to access its environment through the server UI (by clicking on Edge_Agent_1)

*Figure 30: Enlisting an Agent in my Fleet*

The way this works is that every x seconds the edge devices poll the server the same way the administrator does (Portainer API server URL), and they ask if there is any update. If there is, the server responds with the changes that have to be done. In special cases and emergencies, another secure tunnel gets created from the edge to the server (Portainer tunnel server address), giving the administrator real time access.

4. If after an update from the server there is need to pull a new image from the online image repository (dockerhub), this will be done in stage 4
5. In this connection, the administrator will push the image with the software that needs to run on edge (my ML anomaly detection scripts), on the online repository, making it available for the edge devices to find when needed.

# Chapter 5

# 5. Summary

**Objectives Restated**: This thesis aimed to present and compare various anomaly detection methods and algorithms, analyze the importance of the metrics used, and implement the most suitable one for real-time and online fleet in the field.

**Summary of Methodology**: We selected three prominent anomaly detection algorithms: Isolation Forest, Local Outlier Factor (LOF), and Subsequence Anomaly Detection (SAND). We utilized multiple time series datasets, which were preprocessed to ensure clean, normalized, and segmented data. Each algorithm was tested using these datasets, and their performance was evaluated based on key metrics such as precision, recall, F1 score, and processing time.

**Key Findings**: The comparative analysis revealed that the Subsequence Anomaly Detection algorithm consistently outperformed the other algorithms across most metrics, demonstrating high accuracy and robustness. However, it was also noted that SAND had higher computational demands. Isolation Forest was the fastest, making it suitable for real-time applications where speed is critical. LOF offered a balanced performance, suitable for a variety of applications.

**Challenges and Limitations**: During the research, we faced challenges such as difficulties with the MQTT protocol. The webhost we used (vercel) may use a firewall that blocks this messaging protocol so we switched to REST API communication. Another challenge we faced was the cross compilation of the Machine Learning algorithms so they can run on the Raspberry edge devices (aarch64)

**Future Work**: Future work should explore more ways to integrate automation and monitoring techniques. Also, we should enhance the efficiency of edge computing deployments, and develop more robust security measures for data processing. Cross-domain applications and adaptation of these techniques to other fields offer exciting opportunities for further research.

**Conclusion**: This thesis contributes to the field of anomaly detection by providing a comprehensive comparison of three significant algorithms, highlighting their strengths and limitations, and demonstrating their applications in real-world scenarios. The findings underscore the importance of selecting appropriate methods for specific use cases and pave the way for future advancements in this area.

# Bibliography

1. *Anomaly detection in IOT edge computing using deep learning and instance-level horizontal reduction.* **Negar Abbasi, Mohammadreza Soltanaghaei,Farsad Zamani Boroujeni.** May 2024, 2023, SpringerLink.

2. *Data Fault Detection in Wireless Sensor Networks Using Machine Learning Techniques.* **P. Indira Priya, S. Muthurajkumar & S. Sheeba Daisy.** 2022, SpringerLink, p. 2462.

3. *Machine Learning Algorithms - A Review.* **Mahesh, Batta.** 2018, International Journal of Science and Research, p. 6.

4. **Burkov, Andriy.** *The Hundred-Page Machine Learning Book.* s.l. : Andriy Burkov, 2019.

5. **Alexander Zien, Bernhard Scholkopf, Olivier Chapelle.** *Semi-Supervised Learning.* s.l. : MIT Press, 2010. 9780262514125.

6. **Zhou, Zhi-Hua.** *Ensemble Methods: Foundations and Algorithms.* s.l. : CRC Press, 2012. 9781439830055, 1439830053.

7. *Anomaly detection: A survey.* **Chandola, V., Banerjee, A., & Kumar, V.** 2009, ACM Computing Surveys, p. 58.

8. **Varun Chandola, Arindam Banerjee, and Vipin Kumar.** *TR 07-017.* 2007.

9. *A Survey of Outlier Detection Methodologies.* **Austin, Victoria Hodge & Jim.** 2004, SpringerLink, p. 126.

10. **Chaskar, Rushikesh.** Medium. *Understanding Support Vector Machines In Depth Tutorial Part 1 : Linear SVM -Machine Learning Series.* [Online] 9 23, 2020. [Cited: 9 14, 2014.] https://medium.com/that-geeky-guy-codes/machine-learning-svm-easy-indepth-tutorial-4f83442a6d43.

11. **Ravindran, Dhivya.** Medium. *Tree-Based Machine Learning Algorithms Explained.* [Online] 6 19, 2021. [Cited: 9 19, 2024.] https://medium.com/analytics-vidhya/tree-based-machine-learning-algorithms-explained-b50937d3cf8e.

12. **Lim, Yenwee.** Medium. *Unsupervised Outlier Detection with Isolation Forest.* [Online] 3 17, 2022. [Cited: 9 14, 2024.] https://medium.com/@limyenwee_19946/unsupervised-outlier-detection-with-isolation-forest-eab398c593b2.

13. **dey, debomit.** geeksforgeeks. [Online] 6 3, 2024. [Cited: 9 14, 2024.] https://www.geeksforgeeks.org/ml-dbscan-reachability-and-connectivity/.

14. **Amineh Amini, Teh Ying Wah, Hadi Saboohi.** On Density-Based Data Streams Clustering Algorithms: A Survey. *Journal of Computer Science and Technology.* 2014.

15. **Hundley, David.** Medium. *Starbucks Reward Program Project.* [Online] 6 18, 2019. [Cited: 9 14, 2024.] https://dkhundley.medium.com/starbucks-reward-program-project-cfdad91d4779.

16. *Dimensionality Reduction and Visualization in Principal Component Analysis.* **Gordana Ivosev, Lyle Burton, Ron Bonner.** 13, s.l. : ACSPublications, 2008, Vol. 80.

17. **Christopher, Antony.** Medium. *K-Nearest Neighbor.* [Online] 2 2, 2021. [Cited: 9 14, 2024.] https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.

18. **Schulman, Kevin.** The Agitator. *Data Analysis 101: The Z-Score is Your Friend.* [Online] 8 10, 2022 . [Cited: 9 14, 2024.] https://agitator.thedonorvoice.com/data-analysis-101-the-z-score-is-your-friend/.

19. **Paul Boniol, John Paparrizos,Themis Palpanas,Michael J. Franklin.** *SAND: Streaming Subsequence Anomaly Detection.* 2021.

20. **Fei T Liu, Kai M Ting, Zhi-Hua Zhou.** *Isolation Forest.* s.l. : IEEE, 2008.

21. **Markus M. Breunig, Hans-Peter Kriegel,Raymond T. Ng,Jörg Sander.** *LOF: identifying density-based local outliers.* 2000.

22. **Jiawei Han, Micheline Kamber, Jian Pei.** *Data Mining: Concepts and Techniques.* s.l. : Elsevier Inc, 2012.

23. **Kılıç, İlyurek.** Medium. *ROC Curve and AUC: Evaluating Model Performance.* [Online] 9 19, 2023. [Cited: 9 14, 2024.] https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02.

24. **Powers, David M W.** *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation.* Adelaide : Flinders University of South Australia, 2007 .

25. **Stackscale.** Stackscale . *Main cloud service models: IaaS, PaaS and SaaS.* [Online] 02 01, 2023. [Cited: 9 14, 2024.] https://www.stackscale.com/blog/cloud-service-models/.

26. **Beznos, Marta.** n-ix. *Microservices vs monolith: Which architecture is the best choice for your business?* [Online] 1 03, 2023. [Cited: 9 14, 2024.] https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/.

27. **Ojo, Mayowa O.** Medium. *“Dumb” Questions DevOps Newbies Ask: Part 1.* [Online] 1 24, 2024. [Cited: 9 14, 2024.] https://medium.com/@thecloudfairy/dumb-questions-devops-newbies-ask-part-1-84faeb9fa29f.

28. **Famisaran, Patrick.** akcp. *Scaling MQTT Network for Operational Output.* [Online] 8 2021. [Cited: 9 14, 2024.] https://www.akcp.com/blog/scaling-mqtt-network-for-better-operational-output/.

29. **Kang, Steven.** portainer.io. [Online] portainer, 2023. https://www.portainer.io/blog/protecting-portainer-edge-devices-with-cloudflare.

30. **Hui Yie Teh, Andreas W. Kempa-Liehr,Kevin I-Kai Wang.** Sensor data quality: a systematic review. *SpringerOpen.* February 11 , 2020.