Fariha Chowdhury

**Anomaly Detection in Time-Series Data: Analysis on scalability for data-intensive requirements and implementation for industrial scenarios**

Faculty of Information Technology and Communication Sciences

M. Sc. Thesis

November 2024

abstract

Fariha Chowdhury: Anomaly Detection in Time-Series Data: Analysis on scalability for data-intensive requirements and implementation for industrial scenarios

M.Sc. Thesis

Tampere University

Master’s Degree Programme in Computing Sciences (Software, Web and Cloud)

November 2024

In today’s industrial landscape, machines and systems produce vast amounts of time-based data from sensors and monitoring tools. These time-based data are essential for smooth operations and early problem detection to prevent costly disruptions. Real-time anomaly detection is critical in industrial environments for maintaining safety and preventing breakdowns. However, implementing real-time anomaly detection presents several challenges. Due to large-scale data, the system must be robust enough to handle the data storage and processing.

The main goal of this thesis is to explore the best practices to detect anomaly in time-series data. This research is extended to analyse the scalability requirements for a data-intensive industry scenario. Different machine learning models are trained and tested to find out the best one for the specific industry domain. Lastly, a system has been designed and implemented which is used to detect anomaly in real-time and Grafana has been used to notify the user when there is anomaly in the system.

The findings of this thesis are multi-directed. Machine learning models such as Isolation Forest and K-means algorithm have been determined as the best suitable algorithm for anomaly detection in a specific domain. The step-by-step approach – data collection, cleaning and preprocessing to train the models has also been described. The system design and implementation helped to create a tangible outcome of the whole process.

Key words and terms: Time-series data, anomaly detection, sensors, industry machines, machine learning models

The originality of this thesis has been checked using the Turnitin Originality Check service.

# USE OF AI IN THESIS

I have utilised AI tools in my thesis:

No

Yes

The AI tools utilised in my thesis and their purposes are described below:

ChatGPT4

Purpose of using AI tools: To achieve a more formal structure of the document.

Sections where AI tools were used: [List all the steps and sections of your thesis where AI has been applied during your thesis process]

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

Preface

Alhamdulillah. By the grace of Almighty I have reached to the end of my thesis. I am grateful to Jose Martinez Lastra, Luis Gonzalez Moctezuma and Timo Poranen for their guidance during my thesis work. I would like to extend my gratitude to Luis and Timo for helping me in writing my thesis.

Lastly, I would like to thank my parents and husband whose constant support and well wishes have made it possible for me to finish my master’s thesis.

Tampere, 20 November 2024

Fariha Chowdhury

Contents

[1. Introduction 1](#_Toc181887314)

[1.1. Motivation 1](#_Toc181887315)

[1.2. Problem Definition 1](#_Toc181887316)

[1.3. Objective and Scopes 2](#_Toc181887317)

[1.4. Outline 3](#_Toc181887318)

[2. State of The Art 4](#_Toc181887319)

[2.1. Concept of anomaly 4](#_Toc181887320)

[2.2. Publish subscriber pattern 11](#_Toc181887321)

[2.3. Event streaming platform 12](#_Toc181887322)

[2.4. Time series Database 13](#_Toc181887323)

[2.5. Type of scalability 13](#_Toc181887324)

[2.6. Sharding, consistent hashing and partitioning 15](#_Toc181887325)

[2.7. Analysis on scalability of data-intensive requirements 17](#_Toc181887326)

[2.8. Machine Learning Models 22](#_Toc181887327)

[2.9. Evaluation matrices 25](#_Toc181887328)

[3. Methodology 27](#_Toc181887329)

[3.1. Functional and non-functional requirements 27](#_Toc181887330)

[3.2. Selection of the architecture 29](#_Toc181887331)

[3.3. Proposed System Design 30](#_Toc181887332)

[3.4. Containerization 31](#_Toc181887333)

[4. Implementation 32](#_Toc181887334)

[4.1. Components 32](#_Toc181887335)

[4.2. Programming Language and Libraries 33](#_Toc181887336)

[4.3. Data preprocessing 34](#_Toc181887337)

[4.4. Exploratory Data Analysis (EDA) 35](#_Toc181887338)

[4.5. Model Training 37](#_Toc181887339)

[4.6. Combining the components into a project 38](#_Toc181887340)

[5. Discussion and Result 40](#_Toc181887341)

[5.1. LSTM 40](#_Toc181887342)

[5.2. Isolation Forest 41](#_Toc181887343)

[5.3. LOF 42](#_Toc181887344)

[5.4. One class SVM 43](#_Toc181887345)

[5.5. K-means clustering 44](#_Toc181887346)

[5.6. Comparison among machine learning algorithms 45](#_Toc181887347)

[5.7. Grafana visualization 46](#_Toc181887348)

[6. Conclusion 48](#_Toc181887349)

[6.1. Future work 48](#_Toc181887350)

[References 50](#_Toc181887351)

List of Figures

[Figure 1: Anomaly in two-dimensional dataset [4]. 4](#_Toc181887352)

[Figure 2: Collective anomaly in ECG. 5](#_Toc181887353)

[Figure 3: Contextual anomalies in time series data in respect of temperature [4]. 6](#_Toc181887354)

[Figure 4: Anomaly detection techniques in sensor network [23]. 8](#_Toc181887355)

[Figure 5: Real time data processing and anomaly detection workflow in network and storage infrastructure [24]. 9](#_Toc181887356)

[Figure 6: Publish-Subscriber pattern. 11](#_Toc181887357)

[Figure 7: Kafka Producer and consumer. 12](#_Toc181887358)

[Figure 8: Vertical and Horizontal scaling. 14](#_Toc181887359)

[Figure 9: Consistent hashing before and after adding new shard D. 15](#_Toc181887360)

[Figure 10: Kafka topic with partition. 18](#_Toc181887361)

[Figure 11: Red labeled leader partition and blue labeled other partitions. 19](#_Toc181887362)

[Figure 12: Kafka broker read/write operation with replica partitions. 19](#_Toc181887363)

[Figure 13: Partitions consumption by consumer group. 20](#_Toc181887364)

[Figure 14: Multiple consumer groups to handle different tasks. 20](#_Toc181887365)

[Figure 15: Anomaly detection using Isolation Forest [26]. 23](#_Toc181887366)

[Figure 16: Local Outlier Factor [28]. 24](#_Toc181887367)

[Figure 17: Simplified Architecture of the proposed anomaly detection process. 30](#_Toc181887368)

[Figure 18: The given anomaly points of the temperature sensor dataset. 32](#_Toc181887369)

[Figure 19: Dataset first five rows containing timestamp and value column. 35](#_Toc181887370)

[Figure 20: Number of normal and anomalous data in the dataset. 36](#_Toc181887371)

[Figure 21: Month wise min and max temperature distribution. 36](#_Toc181887372)

[Figure 22: Temperature Distribution in dataset. 37](#_Toc181887373)

[Figure 23: Daily mean temperature. 37](#_Toc181887374)

[Figure 24: Docker configuration of the project. 38](#_Toc181887375)

[Figure 25: Workflow of the application. 38](#_Toc181887376)

[Figure 26: Screenshot of table named sensor\_data in TimeScaleDB. 39](#_Toc181887377)

[Figure 27: Confusion matrix of LSTM. 40](#_Toc181887378)

[Figure 28: Detected anomalies in test samples using LSTM. 41](#_Toc181887379)

[Figure 29: Confusion matrix of trained isolation forest model. 41](#_Toc181887380)

[Figure 30: Detected anomalies in test samples using Isolation Forest. 42](#_Toc181887381)

[Figure 31: Local Outlier Factor confusion matrix. 42](#_Toc181887382)

[Figure 32: Detected anomalies in test dataset using LOF. 43](#_Toc181887383)

[Figure 33: Confusion matrix of One-class SVM. 43](#_Toc181887384)

[Figure 34: Detected anomalies in test sample dataset using One-class SVM. 44](#_Toc181887385)

[Figure 35: Confusion matrix of K-means clustering. 44](#_Toc181887386)

[Figure 36: Anomaly detection in test sample using k-means clustering. 45](#_Toc181887387)

[Figure 37: The dashboard of Grafana after data ingestion. 47](#_Toc181887388)

List of Tables

[Table 1: Comparison between TSDB and RDBMS. 13](#_Toc181887389)

[Table 2: Horizontal scaling vs Vertical scaling. 14](#_Toc181887390)

[Table 3: Comparison among sharding, consistent hashing and partitioning. 17](#_Toc181887391)

[Table 4: Comparison among the model results. 45](#_Toc181887392)

List of Symbols and Abbreviations

DB Database

RDBMS Relational Database Management Systems

TSDB Time Series Database

IF Isolation Forest

LSTM Long Short Term Memory

LOF Local Outlier Factor

SVM Support Vector Machine

OC-SVM One-class Support Vector Machine

IIoT Industrial Internet of Things

TP True Positive

TN True Negative

FP False Positive

FN False Negative

1. Introduction

## Motivation

In today’s industrial world, machines and systems are getting more dependent on IoT devices. According to IoT-analytics report [1], it is expected that 18.8 billion IoT devices will be connected by the end of 2024. The report also estimated that around 40 billion of devices will be connected by the end of year 2030. These devices produce a massive amount of data from sensors and monitoring tools which is stored as time-series data [2]. The data is mainly used for data visualization and data mining [3]. This data is important for keeping things running smoothly and catching problems before they cause major issues. Real-time anomaly detection or finding unusual patterns in data as they happen is key to preventing costly breakdowns and maintaining safety.

However, making real-time anomaly detection work effectively is challenging. Time Series Databases (TSDBs) are designed to handle large amounts of time-based data. However, if the overall architecture of the system is not robust enough, then only using TSDB may not be useful enough for spotting anomalies in real-time. On top of that time-series data in industrial scenarios is usually massive due to continuous incoming data from many sensors. Dealing with large-scale data needs special requirements to distribute the data within the system to have a robust result. Using anomaly detection machine learning algorithms online also makes it difficult to maintain a good result. Machine learning algorithms can also be used offline but that is not adaptive to the continuous change of data streams. On top of that, industrial environments have their own unique challenges, like dealing with large volumes of data and needing quick, accurate results.

The reason for this research is to find out the best practices to use TSDBs for real-time anomaly detection in industrial settings and to determine the appropriate machine learning algorithm for a certain industrial domain. This study will look at how to process data, design systems, and use machine learning algorithms to improve how well these detection systems work.

## Problem Definition

The main issue this study attempts to find a suitable way to efficiently perform anomaly detection in real-time in industrial settings. This complex issue involves several major difficulties in effectively handling and processing substantial amounts of time-series data, choosing suitable machine learning methods, and creating a reliable system architecture.

Although TSDBs are efficient at handling huge amounts of time-based data, it might be challenging to use them for real-time anomaly detection. The difficulty lies in processing and analyzing data streams without causing lags or performance problems. This is specially important in industrial settings where prompt detection can avert expensive malfunctions. The configuration and use of TSDBs to satisfy the real-time requirements of industrial applications are explored in this study.

Effective anomaly detection in time-series data requires robust data preprocessing and system architecture. Industrial data often contain noise, missing values, and irregularities. These challenges can reduce anomaly detection accuracy. The large-scale incoming data can make the system lag if proper distributed system architecture is not built.  One of the goals of this research is to design scalable system architectures that ensure efficient anomaly detection.

Accurately identifying anomalies in the frequently complicated and noisy industrial time-series data requires careful selection of machine learning techniques. Determining algorithms that handle massive amounts of data in real-time while balancing processing speed and accuracy is a challenging task. The purpose of this study is to identify the best algorithms for real-time anomaly identification in industrial environments.

## Objective and Scopes

The objective of this research is to find out the answer to the following research questions in respect to anomaly detection in real-time sensor data streaming through a time-series database.

1. What are the best practices (data preprocessing, architectural style) for performing anomaly detection using time-series databases?
2. How can anomaly detection algorithms be scaled to handle large volumes of time-series data in data-intensive industrial applications?
3. What are the most effective machine learning algorithms for anomaly detection in time-series data within the industrial domain?

The research will not cover anomaly detection outside of time-series data or beyond industrial environments. The practical tests will focus on specific industrial scenarios, with findings that may need further research to apply to other fields.

## Outline

This thesis is organized as follows: Chapter 2 provides a comprehensive literature review on time-series databases, anomaly detection techniques in real-time data streaming and scaling the system for large datasets. Chapter 3 outlines the methodology used for anomaly detection and data characteristics. Analysis of building robust system for data-intensive scenarios are also discussed in this chapter. Chapter 4 focuses on the implementation, describing the setup of the database, data ingestion processes, and the anomaly detection algorithms employed. Chapter 5 presents the results and analysis, discussing the performance differences observed among machine learning models and potential areas for further research. Finally, Chapter 6 concludes the thesis with a summary of findings and recommendations for future work.

1. State of The Art

This chapter presents the concept of anomaly and different types of anomalies. Anomaly detection techniques are also described briefly. An overview of the existing research and approaches related to anomaly detection in time-series data has been covered in this section.

## Concept of anomaly

Anomaly can be referred to as an outlier, distraction from normal behavior [4] [5].

A diagram of a number of objects

Description automatically generated

Figure 1: Anomaly in two-dimensional dataset [4].

In Figure 1 there are two large primary clusters N1 and N2. There are also three small clusters o1, o2 and O3. These small clusters are considered as anomaly. This is because a large number of data points fall into the N1 and N2 category whereas a small number of data points does not belong to either of these clusters. Thus, these are outliers.

Types of Anomalies

Time series data can occur in different types of anomalies [6] [7]. Thus, type of anomaly can be divided into several categories.

*Point anomalies:* Point anomaly indicates to single points which are significantly out of normal range. This can happen when a particular point differs from the rest of the points. These are the simplest form of anomalies which are used in anomaly detection.

In Figure 1, o1 and o2 are two different points that deviate from the cluster N1 and N2. O1 and o2 are point anomalies. Point anomalies in fraud detection, sensor malfunction and cybersecurity may indicate critical issues.

*Collective anomalies:* Collective anomalies refer to a sequence of points which are out of normal behavior. Not necessarily, a single point within this sequence is a point anomaly. When looking into the whole sequence of data points collectively it is considered as anomaly.

In Figure 2, the graph refers to an ECG report. The red mark part indicates an abnormal behavior of the sequence. However, a single point within this red mark may not be considered anomaly.

A graph showing a red line

Description automatically generated

Figure 2: Collective anomaly in ECG.

*Contextual anomalies:* Contextual anomalies are also known as conditional or behavioral anomalies. It refers to a data point that deviates from the expected pattern but only within a specific context. Global anomalies are outlier in any context whereas contextual anomalies are considered abnormal only when compared to a certain surrounding or related data.

For example, a sudden temperature drop in summer may seem like anomaly, but it may not be an anomaly in winter.

A graph of time and months

Description automatically generated

Figure 3: Contextual anomalies in time series data in respect of temperature [4].

Anomaly Detection Techniques

There are three main techniques to detect anomalies in datasets. Based on the requirements of the case study any one of these is applied to the datasets.

*Supervised learning:* In supervised learning the training dataset is labeled with normal and anomaly classes [8]. In real life it is difficult to find these types of data as anomaly is kind of rare. Due to this reason the applicability of supervised learning algorithm has limitation.

*Semi-supervised learning:* In semi-supervised learning the training dataset is labeled with normal cases but not with anomaly cases [9]. Thus, the model is trained with only normal scenarios. When it finds something out of normal range it detects it as anomaly.

*Unsupervised learning:* Unsupervised learning is a powerful method that does not require labeled data for training [8]. In real-world applications anomalies are rare and unpredictable. It is also often expensive or difficult to label the anomaly in advance. Unsupervised learning plays a key role in this case.

Anomaly Detection Types

There are several anomaly detection types that have been used for years [10] [11]. Each has its own strengths and weaknesses. Some of these types are briefly described below.

*Statistical approach*:Using statistical algorithms [10] to find anomaly was widely popular in earlier times. The main concept of statistical approach is that the model initially defines a particular distribution. If the data points don’t follow the distribution model or deviates far from it, then the points are detected as outlier. There are two types of statistical approach – parametric and non-parametric methods. Mathematical approaches like Z-score, IQR are used to find the outlier within the data points.

*Distance Based*: In this method the distance among the data points is calculated and the points that are far from the usual ones are considered as anomaly. Common distance-based models are K-Nearest Neighbor (KNN) [12] and DBSCAN [13].

*Density Based*: This method also tries to find anomaly based on data distribution. Density Based method focuses on the density of the data points. Dense region is identified as normal neighborhood whereas points in low-density region are defined as anomaly. Local Outlier Factor (LOF) and Isolation Forest are widely used density-based anomaly detection techniques.

*Cluster Based:* In cluster-based method, anomalies are observed when the data points are not within a range of dense cluster. K-means clustering is a popular method to detect anomalies. The pros of cluster-based technique is that it needs less computing than distance-based techniques. Hence it requires less computation time. The con is that it may not give appropriate result in smaller dataset [14].

*Deep Neural Network:* In recent times neural networks have become more popular in detecting anomalies [15]. Deep neural networks can find complex patterns and relationships among features within the dataset. For effective neural network model, proper data preprocessing and feature engineering is needed.

Anomaly Detection in time series data

Anomaly detection in data streaming was divided into three categories by Lu et al. [4]: offline learning, semi-online learning, and online learning. The study emphasizes that because offline learning cannot adjust to real-time changes in the data stream, it is less suitable for real-time applications. During continuous data production, this causes it to detect new anomalies more slowly.

Offline learning requires batch data to process and train the model whereas this technique requires less computational power and resources. This process has less performance in case of real time data processing. The real time anomaly detection can be harder as it is much slower than online learning. Semi-Offline algorithms are considered as hybrid - where some training is done offline but they are updated periodically to adapt to changes. Notable algorithms include MiLOF [16], NETS, EC-SVM [17] and MCOD [18]. These algorithms balance accuracy with computational efficiency. Online learning is performed specifically where real time data processing is needed. It requires more computational power and resources. The need of online learning is growing as industrial machines require to be monitored in real time. They include osPCA [19], OSHULL [20], VFDT [21] and EFDT [22]. These methods are ideal for dynamic environments but may lack the accuracy of more complex offline algorithms.

In real time anomaly detection full dataset is not available at the same time which makes it difficult to detect the abnormalities in the incoming data. As soon as the data are incoming, a continuous data processing needs to be done and online model learning is required. The paper also identifies ongoing challenges in anomaly detection for data streams, such as improving real-time processing, handling high-dimensional data.

The paper Erhan et al. [23] provides an extensive review of anomaly detection methods specifically tailored for sensor systems in the context of the Internet of Things (IoT). It addresses the inherent challenges of handling big data, energy constraints, and network efficiency while ensuring real-time detection of anomalies. The author emphasizes on efficient data processing of IoT sensor systems due to constraints like energy, computing power, and network bandwidth.

A diagram of a diagram

Description automatically generated

Figure 4: Anomaly detection techniques in sensor network [23].

Riyaz et al. tried [24] to propose a taxonomy in different categories such as machine learning techniques, anomaly detection and big data. The paper surveyed the state of the art focusing on real time anomaly detection in big data.

The paper elaborately described the challenges of real time anomaly detection. Based on the paper, noisy and missing input data, computational cost and inadequate architecture to handle real time big data are the most challenging factors in this case.

Several icons and symbols

Description automatically generated with medium confidence

Figure 5: Real time data processing and anomaly detection workflow in network and storage infrastructure [24].

The Figure 6 depicts the bottom-up sequence of real-time big data processing for anomaly detection, various smart devices communicate via network technologies. These devices generate large amounts of sensor data which are stored in cloud and other storage systems. The datasets collected from sensor devices are then processed using big data processing technologies like Hadoop, Spark, and Apache Storm. The processed data is continuously analyzed and machine learning algorithms are employed to detect anomalies.

The authors, Deshpande and Rahman [25] in this paper justified the the need for an edge-based model of operation and implementation to detect the anomalies in IIoT infrastructure. The vast amounts of data generated in the industries are highlighted as an issue, thus leading to traditional technologies hindering operations because of high latencies, unavailability and even privacy concerns. The study presents a novel approach, which moves data preprocessing, feature engineering and machine learning paradigms to the edge of the system. The paper reports experimental results validating this approach in various industrial applications showing that the speed and accuracy of the complex anomaly detection tasks are considerably higher than the traditional cloud solutions. The analysis also reveals the degree to which these solutions can improve the efficiency and dependability of IIoT systems by performing the data processing closer to where the data is generated, typical of edge computing.

The paper by Yan et al. [26] introduces a novel method for detecting anomalies in time series data. Traditional methods for anomaly detection often struggle with identifying point anomalies related to periodic or seasonal trends in streaming data. To address this limitation, the authors propose DeepAnT, a deep learning-based approach that can detect various types of anomalies including point anomalies and contextual anomalies in both streaming and non-streaming time series data. The paper also discusses the method's limitations due to data quality and contamination.

Ahmad et al. [27] discussed the requirements of unsupervised anomaly detection and proposed a novel technique to determine AD in real time using Hierarchical Temporal Memory (HTM). This method can detect anomalies in a noisy environment as well.

The paper authored by Teng [28] proposed a new nearest neighbor method using existing distance-based anomaly detection algorithm. The new IBAD algorithm introduces a novel scoring system which is calculated from local neighbourhood of the training dataset. The new scores are then used to reshuffle the distances of the data points. Based on the paper, this method has shown better performance in terms of accuracy than classical distance-bases algorithms.

In their paper, Bulla and Birje [29] focuses on the challenges of privacy and communication efficiency in anomaly detection within the Industrial Internet of Things (IIoT). It proposes a federated learning-based framework for detecting anomalies in time-series data. It emphasizes decentralized learning and edge-based computation to ensure privacy and reduce data transmission requirements.

Yan et al. [30] explores deep transfer learning approaches which are highly relevant for industrial time-series anomaly detection where labeled data is often limited. It categorizes different transfer learning methods such as instance transfer and domain adaptation. This makes it easier to apply anomaly detection across multiple industrial contexts. Transfer learning helps models adapt to new datasets without needing extensive retraining. This is considered a significant advantage in dynamic industrial environments.

The study done by Tziolas et al. [31] employs deep learning-based autoencoders such as LSTM and CNN architectures to detect anomalies in time-series data from the elevator industry. It illustrates the effectiveness of deep learning in improving anomaly detection in complex, multivariate industrial datasets, overcoming many limitations of traditional statistical methods.

## Publish subscriber pattern

Publish subscriber is a messaging pattern to communicate and exchange data in between sender and receiver. It is common to use this pattern in real time data streaming scenarios such as monitoring IoT devices.

The publisher publishes the message and categorize the topic based on the content. The publisher has no information about the subscriber. The receiver subscribes to the specific topic of its interest and waits for the data. When the publisher publishes the data of that topic the subscriber receives the data.

A white rectangular object with black text

Description automatically generated

Figure 6: Publish-Subscriber pattern.

Topic is the channel that is used by the publisher to send categorized messages. A subscriber can subscribe to multiple topics. Also, one topic can be subscribed by multiple subscribers.

Message broker is an intermediate component in between publisher and subscriber. It manages the messages sent by the publisher and routes them to the specific subscriber. It also helps to scale the system when large number of messages are published in a short period of time.

## Event streaming platform

Event streaming platform helps to stream data as real time incoming data. In real life scenarios, IoT devices or sensors keep sending data to the system in a large scale. This data is passed through an event streaming platform so that data storage and data analysis can be done in real time. In this case, event streaming platform will pass the data in a sequential manner in respect of time and send data through a machine learning model to detect whether there is an anomaly or not.

Apache Kafka is an open-source project which was originally developed by LinkedIn. It is widely used to stream real time data from databases, sensors and cloud services. It enables high-throughput and low-latency data streaming which is ideal for building real-time data pipelines and applications. Kafka organizes data into topics and topics are further partitioned for concurrent processing. Producers write to these topics, while consumers read from them, ensuring scalability and fault tolerance. Common use cases include real-time analytics, event sourcing and microservices communication. Kafka's ecosystem includes tools like Kafka Connect for integrating with external systems.

A diagram of a cluster

Description automatically generated

Figure 7: Kafka Producer and consumer.

Kafka Streams is an adaptable library designed for creating highly efficient, flexible and redundant stream processing applications over Apache Kafka. It enables the encoding and execution of multiple data streams which above all, their source is begun by Kafka topics and allows a system to transform, aggregate or enhance the data in motion. Kafka Streams is intended to be low weight, user friendly and an extension to the messaging system. It is very suitable for real-time analytical applications, real-time monitoring or complex event processing applications.

## Time series Database

Time series databases are considered more efficient than relational databases when dealing with time series data. Time series databases can handle large datasets efficiently by scaling and partitioning the data tables in real time. One of the main benefits of a TSDB is its ability to easily handle time-based queries - finding data within a certain time range. In a relational database, these tasks require more complex queries which can slow things down as the amount of data grows.

TSDBs also use better methods to compress and store data. Compression helps reduce storage space for data that is repeated, such as time series data. Long-term data storage is made easier to manage with features like automatically deleting or summarizing obsolete material. While relational databases can store time-series data, they are not as efficient for this purpose. For applications like IoT, system monitoring, financial data, and environmental tracking, time-series databases are a more effective choice because they are optimized specifically for time-stamped data.

In Table 1, comparison between Timeseries DB and relational DB has been described broadly. Both have its own advantage and disadvantage. It is users’ responsibility to find out the need based on the requirements and select the feasible one.

|  |  |  |
| --- | --- | --- |
| **Compare** | **Timeseries DB** | **Relational DB** |
| Data Type | Time series data | Structured data in tables |
| Query Language | SQL like language | SQL |
| Scalability | Horizontal | Vertical |
| Partitioning | Native | Manual |
| Storage Optimization | Compression and data pruning for time-series data | Optimized for structured table-based storage |
| Examples | InfluxDB, TimescaleDB, Prometheus, OpenTSDB | MySQL, PostgreSQL, Oracle, SQL Server |

Table 1: Comparison between TSDB and RDBMS.

## Type of scalability

An application can be scaled in two ways – horizontal scaling and vertical scaling. Both types have their own advantages and disadvantages. Moreover, based on the system needs, either horizontal or vertical scaling is implemented. Some application may use both as well.

**Horizontal scaling** is theincrement or decrement of number of nodes in a cluster or system based on the workload. For example, adding virtual machines when the workload is high and removing when workload is low.

A screenshot of a computer

Description automatically generated

Figure 8: Vertical and Horizontal scaling.

**Vertical scaling** increase or decrease number of powers such as server, CPU to handle the fluctuating workload.

|  |  |  |
| --- | --- | --- |
| **Compare** | **Horizontal scaling** | **Vertical scaling** |
| Definition | Increasing nodes in a cluster or system | Increasing power of the system |
| Example | Adding Virtual Machine | Adding CPU, server |
| Load Balancing | Necessary in distributed system | Not required |
| Implementation | Easer to implement | Harder to implement |
| Downtime | No | Yes |
| Failure tolerance | Yes. Other clusters can give backup of one fail | No in case of single unit |
| Maintenance | Higher | Lower |
| Performance | Higher | Lower |
| Limitation | Higher number of nodes can be added | Limited amount of resource |

Table 2: Horizontal scaling vs Vertical scaling.

## Sharding, consistent hashing and partitioning

Data are usually stored in databases. For time series data it is ideal to choose a time series database. Time series database is better performant in respect of RDBMS while working with time-series data. The reason behind it is mainly the scalability of the database. Time series database can also be scaled in both horizontal and vertical way.

In horizontal scaling database can be scaled up by using *sharding*. Sharding is a partitioning method by which larger database is divided into smaller parts and distributed to multiple nodes or servers. These smaller parts are called shards. By doing this the performance of the database can be improved and will gain the ability to handle high traffic of data.

*Consistent hashing* is another technique which distributes the data among the shards uniformly. The goal of this method is to minimize the number of data movement when shards are added or removed from the system. The requests and the shards are placed in a virtual ring named hashring. The new shards are placed in the ring based on hashing algorithm. The nearest data are shifted to the new shards. In this way the other data and shards are kept unaffected. Same way is applicable when a shard is removed from the ring. The nearest shards get the data from the removed shard.

The whole process can be described in a more elaborate manner. For instance, there are three shards A, B and C. All the data are distributed among these three shards. Now due to high traffic a new shard has been created. The hashring will place this new node in the ring based on the hashing algorithm. Let’s assume that D is placed in between B and C

A screenshot of a computer screen

Description automatically generated

Figure 9: Consistent hashing before and after adding new shard D.

shard. So now B and C will share their data points with D while A will remain same as before. In this way, it will keep other parts unaffected and make sure the minimum movement of the data among shards.

**Partitioning**

*Partitioning* refers to dividing a dataset into smaller, manageable segments or "partitions" to optimize storage, access and query performance. Each partition typically holds a subset of the data and can be distributed across multiple nodes or servers.

Various partitioning methods are available, each with unique benefits depending on the specific needs of the system, data distribution and access patterns. Some commonly used partitioning techniques is described below.

*Range Partitioning:* Data is segmented into partitions according to specific value ranges. For example, user records might be partitioned based on user IDs: user IDs 1–1,000 go to Partition A, 1,001–2,000 go to Partition B, and so on.

*Hash Partitioning:* A hash function is applied to the partition key (e.g., user ID, product ID), and the result determines which partition the data belongs to. This helps distribute data more evenly across partitions.

*List Partitioning:* Data is divided based on specific values. For example, if you’re storing country-specific data, you might partition based on the country code.

*Composite Partitioning:* This combines two or more partitioning methods, such as hash and range partitioning, to fine-tune how data is distributed.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Feature** | | |  | | --- | | **Sharding** | | |  | | --- | | **Consistent Hashing** | | |  | | --- | | **Partitioning** | |
| Goal | Split data across independent databases | Efficiently map data to nodes with minimal reshuffling | Optimize query performance and storage management |
| Data  Distribution | Manual or custom-defined | Automatic based on hash function | Based on predefined partitioning rules |
| Data  Movement | Requires rebalancing when adding/removing shards | Minimal movement when nodes change | Movement depends on partitioning strategy |
| Use Cases | Large-scale databases, microservices | Distributed caching, key-value stores | Database optimization, query performance |
| Challenges | Cross-shard queries, rebalancing | Uneven distribution, complexity | Hotspots, rebalancing, complex queries |

Table 3: Comparison among sharding, consistent hashing and partitioning.

## Analysis on scalability of data-intensive requirements

The industry sensors generate thousands of data which can be temperature, sound, humidity, vibration and many others in seconds and to cope up with this frequency a robust system is needed. The system must be scalable, fault tolerant and have the capability to handle large scale data. Even though implementation of the scalable application has not been done in this research work, analysis on how to build such a system will be discussed in this section.

To build a scalable system, each component of the system must have the capacity to handle large scale data. The components such as event streaming platforms, data storage, data processing and transformation must be scalable. The distributed architecture such as microservices can be used to maintain the segregation within the components.

**Data Ingestion**

The first step of an application is to gather data from the sources. In IoT scenarios, the sources are mainly thousands of sensors which keep generating data continuously. To get the data from the sources and process it in real time the data collecting points must be scalable enough.

Apache Kafka is considered high-throughput, horizontally scalable and fault tolerant system which is widely used in real time data processing scenarios. In Apache Kafka, each component such as producer, consumer, broker and topics can be scaled to handle large scale data.

Topics are the message streams in Kafka. There can be one or multiple partitions in each topic. The partitions are ordered. When the load of messages rise, topics can be partitioned further to give space to the newly added messages. Each message within the partition is given an incremental id named offset.

A group of colorful rectangular objects

Description automatically generated

Figure 10: Kafka topic with partition.

Kafka brokers are responsible to forward the message streams to the consumers. Thus it is necessary that brokers are scalable too so that it can handle large data streams. Kafka architecture makes sure that brokers are scalable enough. There are multiple brokers within Kafka cluster that are ready to receive message streams. By connecting to one broker, one gets connected to the whole cluster. Kafka is a fault-tolerant system. Partitions are replicated more than once so that if one broker fails others can forward the message to the consumer.

To have a better understanding of the replication process, let’s assume that there are three brokers in a Kafka cluster. Topics are divided into partitions and each broker contain a copy of partitions. There is one leader among these partitions and others just keep the synchronized message with the leader. The leader is managed and elected by another library named zookeeper. Zookeeper keeps the metadata stored which helps the producer and consumer to get information about the leader of the partition. All read and write operation is done through the leader partition and others keep syncing with the leader. If for any reason leader fails, then another leader is elected from the remaining partitions and the work goes on. By building a distributed system Kafka makes sure that the system is fault tolerant.

A screenshot of a screen

Description automatically generated

Figure 11: Red labeled leader partition and blue labeled other partitions.

A screenshot of a diagram

Description automatically generated

Figure 12: Kafka broker read/write operation with replica partitions.

Kafka consumer can be grouped together for any specific topic to have a higher throughput. The group share a unique group id. Within a consumer group each consumer read data from a unique partition. If there is only one consumer within a group, then the consumer will receive message from all the partitions. However, if there are two or more consumers within the group, then it will receive message from different partitions. If there are more consumers then the partitions, then the unassigned consumer will remain idle. That’s why it is important to maintain the number of partitions and consumers.

A close-up of a computer screen

Description automatically generated

Figure 13: Partitions consumption by consumer group.

There can be multiple consumer groups within Kafka system. It is helpful when parallel processing is needed for the same topic. For instance, after receiving the topic, one consumer group can be used to store data to the database and another consumer group can be used for real time data analysis. In IoT systems, the data streams are frequent and large. If multiple producers are used to read the data, then it is necessary to maintain the scalability of the consumer as well. Otherwise, the system will lag when trying to write the data to the consumer. When there is different consumer groups present to handle different tasks, the system will be able to write the data without any issue.

A diagram of a diagram

Description automatically generated

Figure 14: Multiple consumer groups to handle different tasks.

To scale Kafka consumer for data consumption from a topic is to add more consumers within the consumer group. The higher the consumers in a group the higher the latency output of the consumer group. It is needed for writing to database and to do complex computation that takes more time. It is also important to not keep consumers more than needed. If there are more consumers than the number of partitions, then the consumer will sit idle which is a loss of resources. To calculate this number of partitions and consumers must be balanced.

If needed, vertical scaling can also be obtained in Kafka consumer. It is possible that only horizontal scaling is not enough when the system handles large scale data with complex calculations. In that case, multi-threading model can be a good option. In single-threading model, only one thread works to fetch data from the poll and then does the processing. With multi-threading model, each data can be processed by different threads which significantly increases the efficiency of the system. However, multi-threading model does not guarantee the order of the messages as data from same partition may get processed parallelly. Also, it is only beneficial when other system dependent on the model can handle the load.

**Distributed architecture**

Microservice architecture is a distributed architectural style that structures an application as a collection of small services. Each of this service is running in its own process and communicate through lightweight methods like HTTP APIs or messaging queues. In monolithic architectures all components are interwoven and deployed as a single unit. For creating a large-scale application microservices offer several advantages that address the complexities and challenges associated with handling vast amounts of data.

Each microservice can be scaled independently based on its load and resource requirements. For data-intensive applications, services responsible for data ingestion, processing or analytics can be scaled horizontally to handle increased data volumes. In this process it does the work without over-provisioning other parts of the application. By scaling only the services that need it, it is possible to optimize resource usage and reduce costs. This is crucial when dealing with large datasets that require significant computational power.

In a microservices architecture, a failure in one service doesn’t impact the entire system. Rather the specific service is affected and others run as usual. This isolation is critical in large-scale data applications where components may fail under heavy load.

Microservices can be designed to handle specific stages of data processing such as collection, transformation, analysis and storage by creating a seamless data pipeline that can be scaled and optimized at each stage.

## Machine Learning Models

Several machine learning algorithms have been studied that are efficient in detecting anomalies in industrial scenarios. Different algorithms have different strengths and weaknesses. The algorithms that are more relevant to the use case that has been researched on this topic is discussed below.

**2.7.1 LSTM**

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN). It is designed to overcome the limitations of traditional RNNs. RNNs are well-suited for time-series and sequence-based data. However, the downside of RNN is that it often struggles with learning long-term dependencies due to the vanishing gradient problem. LSTMs address this by introducing three different gates [32]. These gates control the flow of information in and out of the cell state, allowing the network to remember information over longer periods.

LSTM consists of three gates: forget gate, input gate and output gate. The purpose of the forget gate is to decide what information can be discarded from the cell state. The job of the input gate is to modify the cell state by adding some new information. Lastly, the output gate determines what information should leave the cell state.

LSTM is an unsupervised machine learning language model. Thus, there is no need to label the training dataset. It is useful in detecting anomaly detection as it learns the pattern itself from the training dataset and if any data point ranges out of the usual the model considers that as outlier. LSTM is also effective for anomaly detection in large scale dataset due to its learning long term pattern.

With many advantages, LSTM comes with some disadvantages as well. The computation cost is expensive and training time is longer. It also requires fine tuning of parameters which is complex to learn.

**2.7.2 Isolation Forest**

The Isolation Forest algorithm, as an unsupervised machine learning approach, is specially made for detecting anomalies. It does this by isolating the data points by repeatedly dividing them into partitions until each point is placed into its own partition characterized by a quorum [33]. This is the number of partitions necessary to capture a point; this number for the anomalies is lower than that of normal points. Traditional approaches aim to understand the structure or model of normal data first and then label everything that deviates from it as an outlier. In other words, IF simply distinguishes outliers by their rarity as opposed to everything else in the dataset. Figure 15 shows how the tree is generated by the model and finds the anomaly.

A diagram of a forest tree

Description automatically generated

Figure 15: Anomaly detection using Isolation Forest [26].

This approach is particularly advantageous with big data due to its low resource consumption and ability to perform analysis in real time. IF is also effective with data that have numerous dimensions since it splits the data by picking a few features at random. It is adaptable for various types of data because it also accommodates different forms of data including continuous and categorical data.

**2.7.3 Local Outlier Factor (LOF)**

Local Outlier Factor (LOF) measure the local density deviation of a data point compared to its neighbors [34]. Points in regions with significantly lower densities than their neighbors are considered outliers.

A diagram of a number of red circles

Description automatically generated

Figure 16: Local Outlier Factor [28].

The advantage of LOF is that it is an unsupervised method specifically designed to detect anomalies. LOF is versatile and can be adapted to various kinds of data and applications for detecting anomalies.

**2.7.4 One class SVM**

The One-Class Support Vector Machine (OC-SVM) focuses on the unsupervised novelty detection and is a sub-type of the famous Support Vectors Machine algorithm. It encapsulates the majority of the normal instances in such a way that any other point which lies outside of this encapsulation is considered as an outlier [35].

This is done by applying certain mathematical functions known, for example, kernel functions like RBF or linear to the data and projecting the tri-dimension data into a higher dimensional space to find a plane (hyperplane) which has the biggest distance between the normal data points and a margin that contains no data points. The function takes as input new points and determines whether they are categorically inside (normal) or outside (anomalous) the cut. This involves the use of parameters such as nu, which affects the number of support vectors and the number of outliers expected, and gamma, which determines point influence, to achieve that end.

OC-SVM finds use in the situations where there is large majority of normal data with the exception of outliers that need to be detected. It is also suitable for high-dimensional data since kernel methods can be used to cope with complex normal behavior. It, however, demands careful tuning of parameters. The values of nu and gamma vary its efficiency. Also, there is a need for a very high volume of clean training data that represents the normal state accurately for it to learn the normal characteristics effectively. In addition, it could be useful in circumstances where there is a high level of noise or normal and abnormal data distributions have a high level of overlapping.

**2.7.5 K-means clustering**

K-means is a method of grouping data which helps in creating k number of clusters and each point is assigned to the cluster whose center is closest to it, known as its central value. The objective is to arrange the data in such a way that the objects are similar within the cluster but as different as possible with the objects of other clusters. This technique has several applications including but not limited to customer segmentation, image compression and pattern recognition.

The way the algorithm works carries different steps. The first step is to take k different centroids randomly. The next step is to assign each data point to the closest one of the k centroids in practice usually the Euclidean distance is used to perform this operation. The next step is to find out new centroids for each of the clusters containing different data elements. This process in carried out until the centroids do not change or some fixed number of repetitions is achieved.

K-means is not only easy to implement but also able to produce satisfactory results with large amounts of data. Nevertheless, the methodology lying behind K-means is not free of defects. One must define the number of clusters to group the data before the algorithm can be executed, which in reality is often difficult to ascertain. It is also affected by the limits of the architecture been used in that if the cluster center changes k-means gives a k-means outcome. A second problem is that K-means does not operate well in the presence of noise since the presence of outliers affects the centroid and can move it towards the outlier. This geometric hypothesis often does not hold true. K-means tends to be inefficient because it also assumes evenly sized clusters with semi-spherical shapes.

## Evaluation matrices

The confusion matrix is a useful tool for analyzing how effective a given classification model is. It is a matrix that compares the actual categorization of an object with the one that is predicted and helps to assess the performance of a model.

*True Positives (TP):* These instances correspond to the actual class as positive and the class predicted by the model is also positive. The model was able to correctly detect positive cases.

*True Negatives (TN):* These instances correspond to the actual class as negative and the class predicted by the model is negative. The model was able to correctly detect negative cases.

*False Positives (FP):* These instances correspond to the actual class as negative but the model predicted it as positive. The model misclassified these negative cases to positive.

*False Negatives (FN):* These instances correspond to the actual class as positive but the model predicted it as negative. Such cases were not detected by the model, as they were presented as negatives.

*Accuracy:* The number of correct predictions made in relation to the total number of predictions made.

Accuracy=

*Precision:* Precision is a measure of how many of the instances that the model predicted as positive (true) are actually positive. It focuses on the quality of positive predictions. High precision means that when the model predicts a positive outcome, it is usually correct. It is calculated as:

Precision=

*Recall:*Recall measures how many of the actual positive instances were correctly identified by the model. It focuses on capturing all relevant positive cases. A high recall indicates that the model successfully identifies most positive cases. It is calculated as:

Recall=

*F1 Score:*The F1 score provides a balanced measure when there is an uneven class distribution. It is the harmonic mean of precision and recall. It combines both precision and recall into a single metric to give a more holistic view of a model's performance. It is used particularly when false positives and false negatives are both important to minimize. It is calculated as:

F1 Score =

1. Methodology

This chapter outlines the functional and non-functional requirements of the system, design choices, methodologies and techniques used in the development of the system for detecting anomalies in time-series data. It covers the strategies for scaling the solution to handle data-intensive industrial applications. Each step in the system's design is discussed in detail highlighting the rationale behind key decisions and their impact on the overall performance and scalability.

## Functional and non-functional requirements

Functional requirements specify the functions of the system that must be rendered in the system to achieve the user goal. They are aligned towards the actions and workings of the said system. The functional requirements of the real-time anomaly detection application are listed as follows.

*Real-Time Data Ingestion:* The system must have the capability of gathering and processing data on an ongoing basis and in real time from multiple data sources.

*Anomaly Detection:* The system is expected to notify users of the presence of unusual features or behaviors in the data being received upon analysis. The few minutes or seconds spent by the detection process should not go beyond this limit.

*Data Storage and Management:* The system should be able to design and implement a storage schema that handles time-series data in enormous volumes satisfactorily. It should support effective query responses and retrieval of both live and past data without undue time lags.

*Data Streaming and Processing:* The system should make allowance for the constant data stream processing in the system such that there are no or minimal breaks from ingestion to storage to processing of the data.

*Scalable Data Handling:* The system should be able to accommodate increasing amounts of data by either growing the system or by smartly partitioning and distributing data across the system.

*Data Visualization and Alerting:* The system must allow the users to see the incoming data and the outliers present in the current data stream and provide alerts about the same. It should have a very robust alerting system that can detect and send alerts to the users in the instance of an anomaly occurrence or a system crash.

*Inter-component Communication:* The system should support communication between the different components in a loosely coupled and asynchronous manner. This helps to promote good data flow between the components and allow off scaling of individual components.

Non-functional requirements define how well the system performs its functions, focusing on the system's operational qualities such as performance, scalability, security, and usability.

*Scalability*: The system must be scalable both vertically and horizontally to handle increased workloads and data traffic without performance degradation.

*Performance*: The system should ensure low latency in processing data and detecting anomalies. High-throughput capabilities are necessary to handle large-scale data flows without causing bottlenecks.

*Fault Tolerance and Reliability*: The system must be fault-tolerant, ensuring that if one component fails, the others continue to operate without data loss or significant downtime. Redundancy mechanisms should be in place to ensure system reliability.

*Availability*: The system must ensure high availability, operating continuously with minimal downtime. This is crucial for real-time data processing systems that cannot afford interruptions in service.

*Security*: The system should maintain secure data handling practices. This includes ensuring data confidentiality, integrity, and authorized access control across all layers. Secure communication protocols should be in place to protect data in transit.

*Maintainability*: The system should be easy to maintain, update, and extend. Its architecture should support smooth upgrades and modular changes without affecting other parts of the system.

*Data Integrity*: The system must ensure that data is not corrupted or lost during transmission, storage, or processing. Proper mechanisms should be implemented to always guarantee the consistency and correctness of the data.

*Energy Efficiency*: The system should optimize resource usage, particularly if operating on constrained devices such as IoT sensors, or when deployed in edge computing environments. It must aim for energy-efficient processing and storage of data.

*Usability*: The system should be user-friendly, with a clear and intuitive interface for monitoring, visualizing data, and managing alerts. The complexity of the backend system should not impact the ease of use for the end-users.

*Latency and Throughput*: The system should minimize delays in data processing and ensure high throughput, capable of processing large amounts of data in real-time with minimal latency to meet industrial operational demands.

## Selection of the architecture

After analyzing the functional and non-functional requirements of the application the next step is to choose the appropriate architecture for the application. To choose the architecture, multiple architectures were initially studied and analyzed. By thoroughly investigating the architectures, finally event-driven architecture has been chosen to be applied to this application.

Event driven architecture is well-suited for applications like anomaly detection where low latency and real time data processing is needed. It is able to do data processing asynchronously. It can react to incoming data instantly which will be helpful for timely detection of anomaly.

Event streaming tools can be easily integrated within this architecture. It is also easy to scale horizontally by adding more nodes – topics, consumers and partitions to the system when high-velocity data streams are coming to the application. Due to its high scalability, this architecture is suitable for the application.

Fault tolerance of event driven architecture is also another reason of choosing this for the application. If one service fails, the others can keep working. The event broker keeps storing the incoming data until the service is back online. This way it maintains the fault isolation of the system.

## Proposed System Design

The design of the application can be divided into two core parts – data storage and anomaly detection of incoming real time data.

A diagram of a data stream

Description automatically generated

Figure 17: Simplified Architecture of the proposed anomaly detection process.

The Figure 17 depicts the architecture of the proposed system design. The data is first streamed through data streaming tool simulating it like a real time data streaming. As data is ingested by event streaming platform it is then consumed by the subscriber. Subscriber subscribes to the topic beforehand. When subscriber receives the data, it will first store the data into time series database. Time series data is chosen because it is efficient for large scale time series data. Alongside, consumer will send the incoming data for anomaly detection through a pretrained machine learning model. This model has been trained beforehand to recognize normal patterns in the data. Its purpose is to determine whether the incoming data deviates from these patterns. The model will process the data and decides whether there is an anomaly. Then the result will be stored back to the database for future analysis. A visualization tool will be connected to the database in real time. The tool will fetch the data from the database continuously. The user will be able to monitor the system in real time. If there is an anomaly in the data, then the tool will be configured in such a way that it will create an alert.

## Containerization

It is also decided to implement containerization into the application. Containerization will enable to package application code along with all its dependencies, configuration files and libraries into a single unit called a container. This approach solves compatibility issues that often arise when moving software between different computing environments.

Another reason to use containerization is that it is lighter than traditional virtualization. Unlike traditional virtualization, containers do not require a full operating system within each instance. Instead, they share the host machine's OS kernel which is more lightweight and efficient. This reduces overhead and allows faster startup times. It helps to enable more containers to run on a single machine compared to virtual machines.

2. Implementation

In this chapter the components of the systems and the integration among the components is described. The machine learning models that retrained with the dataset are also discussed elaborately to give a better understanding of the algorithms.

## Components

**4.1.1 Dataset**

The dataset for this implementation has been chosen from Numenta Anomaly Benchmark (NAB) datasets. This dataset is a real world known case named *machine\_temperature\_system\_failure.csv* under this NAB datasets [36]. The dataset provides temperature sensor data of an industrial machine. The first anomaly of this dataset was a planned shutdown. The cause of second anomaly is unknown and difficult to detect. The reason of the last anomaly case is also unknown. Figure 18 shows the whole dataset along with anomaly datapoints marked in red.

A graph showing a red and blue line

Description automatically generated

Figure 18: The given anomaly points of the temperature sensor dataset.

**4.1.2 TimeScaleDB**

For data storage TimeScaleDB has been used in the application. TimeScaleDB is an extension of PostgreSQL which is efficient for time series data storage and analysis. It is an open-source database which is highly scalable for time-series data. It uses SQL as query language unlike other time series databases where a new query language is needed.

TimescaleDB introduces features like automatic partitioning of data based on time called "hypertables". It allows efficient querying, storage and analysis of large time-series datasets. It also supports advanced time-series functions, continuous aggregations and real-time analytics. This is well-suited for handling high-velocity data and performing tasks like anomaly detection, trend analysis and forecasting.

**4.1.3 Grafana**

Grafana is a visualization and analytics software that is used in the application to monitor real-time data from various sources. It stores them for a long period of time for querying and visualization. It features an extreme control and flexible design for building presentation dashboards. In addition to that, it enables the user to view metrics and logs, create and manage alerts, and keep track of the hosted systems.

**4.1.4 Docker**

For containerization and deployment, Docker has been used in this application. Docker is a free and open-source version that provides developers with tools to automate the working application’s deployment and leverages the concept of container. This includes containers viewed as the smallest, self-sustaining and portable pieces of software that can be run. It contains everything which is essential to execute an application such as program codes, runtime, libraries, and system tools. This allows the application to be executed uniformly in all available environments.

## Programming Language and Libraries

*Python:* The entire application has been built in the Python programming language. The machine learning models also were trained with python and python provided libraries. Machine learning which is implemented in python is predominantly the most utilized programming language because of its straightforward implementation.

*Scikit-learn:* Scikit-learn is a Python-based free software machine learning library. It provides accessible high-quality data analysis and data mining tools as well as machine learning. Scikit-learn was frequently used during model training. It is well established in areas such as classification, regression, clustering and model selection and preprocessing.

*NumPy:* The programming language Python comes with dedicated libraries such as NumPy which assist in working with matrices and n-dimensional vectors. This free library contains several code libraries applied to do math on the given structures. It is a very important tool for all the libraries which require numerical methods.

*Pandas:* Pandas is a data manipulation and analysis library and is implemented in the python programming language.

*Matplotlib:* Matplotlib is a graphical library for python which is applied for uses such as creating interactive graphs to aid visualization. In machine learning, it helps to depict the distribution of data points. All the graphs in this thesis paper were created using Matplotlib.

*Keras:* Keras is a high-level neural network API, written in Python, that can work on top of Tensorflow. It is used for building the basic building blocks for a neural network. Its design is fast, simple and powerfully effective. Keras was mainly used for LSTM training in the application.

*Joblib:* Joblib was very often used in the application to save and load the training models of machine learning or other heavy data structure. It is a library from Python with optimizations for work with large arrays, particularly NumPy one.

## Data preprocessing

Data preprocessing is an important part during model training with machine learning algorithm. It transforms raw data to a suitable format for training models. Proper data preprocessing can even improve the accuracy and performance of the trained model. There are several steps that has been implemented in data preprocessing step that are described below.

*Data Cleaning*: In incoming data, there can be missing values. It is necessary to either remove the missing values or fill the values with mean values of that column. For the selected dataset it was checked whether there are any missing values, however, there was no missing values.

*Feature Scaling:* Min-max scaling is used to scale data to a specific range [0,1]. This ensures that features with larger magnitudes don't dominate the training process. All the features of the selected dataset were scaled using min-max scaling in the application.

*Data Splitting:* Data was split into two parts – training set and test set. Training set contains 80% of the data and test set contains 20% of the data. The training set is used to train the models. And the test set is used to assess the performance of the model.

## Exploratory Data Analysis (EDA)

In this part, the dataset is analyzed in different perspective to have more in depth knowledge about the dataset. Exploratory analysis helps to find out hidden patterns and trends within the dataset if there is any.

In Figure 19, the first five rows of the dataset are displayed. There are only two columns in this dataset - timestamp and value. The timestamp column is a time-series data column which shows data in five minutes interval. The value column is a numerical data showing the temperature at a certain timestamp.

**A screenshot of a black screen

Description automatically generated**

Figure 19: Dataset first five rows containing timestamp and value column.

From the Figure 20, it can be identified that the dataset is imbalanced as the number of anomalous data is rare in respect of normal data points. In real life scenarios, anomaly in machines are rare which results in small number of anomaly data points. In this dataset, the total number of data points are 22,695. The number of normal data points are 20,427. whereas anomalous data points are only 2,268.

**A blue and red rectangular graph

Description automatically generated**

Figure 20: Number of normal and anomalous data in the dataset.

In the dataset there are data from three different months – Dec, Jan and Feb. The minimum and maximum temperature in monthly basis is also analyzed to determine if there is any trend or pattern. In Figure 21, it is displayed that the min temperature in December is a bit over 0 degree. From the figure it can also be noticed that the max temperature in all months is consistent measuring over 100 whereas the minimum temperature has varied a lot in these months.

A graph of a bar chart

Description automatically generated with medium confidence

Figure 21: Month wise min and max temperature distribution.

From the temperature distribution graph in Figure 22, it is noticed that temperature frequency is mostly in between 80 to 100 degrees Celsius. The temperature below 60 and above 100 is significantly lower than the temperature in between 80 and 100.

**A graph of a distribution of machine temperature values

Description automatically generated**

Figure 22: Temperature Distribution in dataset.

The daily mean temperature also gives some insights of the dataset. There are clear certain drops and spikes in the graph which may happened due to malfunction in the machine.

A graph showing the temperature of the day

Description automatically generated

Figure 23: Daily mean temperature.

## Model Training

After data preprocessing and exploratory data analysis, models that are good at anomaly detection have been chosen for training. The models that are trained for this dataset are – Isolation Forest, LOF, LSTM, One-class SVM and k-means clustering. Each of these models has shown its strength and weakness which is discussed in result section.

## Combining the components into a project

All the components and tools mentioned above have been used to create the entire application. The whole application was containerized with Docker for a smooth deployment process. Now the application can be run with a single command in any local machine.

A screenshot of a computer

Description automatically generated

Figure 24: Docker configuration of the project.

*Kafka-producer:* In application, the test part of dataset is configured in Kafka-producer. The topic name is ‘sensor-data’. The data rows are flushed from producer in 2ms interval from a csv file.

A diagram of a flowchart

Description automatically generated

Figure 25: Workflow of the application.

*Kafka-consumer:* On the other hand, Kafka-consumer is listening to ‘sensor-data’ topic. Whenever, it starts getting data from the topic it does two things. At first it saves the data to the database. And simultaneously it sends the data through the machine learning model that is already set in the application. After getting the result, whether there is anomaly or not the result is also stored into the database. This loop is running until producer keeps sending the data.

*TimeScaleDB configuration:* The database is configured before running the application. In TimeScalDB, hypertable is used for time-series data storage. A hypertable named ‘sensor\_data’ has been created for this purpose. There are three columns in the table – timestamp, value and is\_anomaly column.

A screenshot of a computer

Description automatically generated

Figure 26: Screenshot of table named sensor\_data in TimeScaleDB.

*Grafana Setup:* Grafana was configured with ‘sensor\_data’ table beforehand. The configuration process is easy using this tool’s UI. A new dashboard was created based on the need. In dashboard, there is only one line graph and two gauges. The line graph gets data in real time form TimeScaleDB table. When the column ‘is\_anomaly’ is true, this can be easily detected from the Grafana dashboard.

1. Discussion and Result

In this section, the outcomes of the machine learning models for anomaly detection in time-series data is displayed. A comparison among this models’ output is also discussed in the later part of this section.

## LSTM

In Figure 27, the confusion matrix of LSTM model clearly indicates how the model performed in detecting anomalies in the given dataset. The matrix consists of four principal components, which are true positives (377) where the system has detected the anomalies correctly, true negatives (3866) where the normal data points have been correctly classified, false positives, (76) in which case normal points were incorrectly regarded as anomalies, and false negatives (190) where the system missed classifying the anomalies and termed them normal and defensive.

**A blue squares with white text

Description automatically generated**

Figure 27: Confusion matrix of LSTM.

In Figure 28, the graph of test dataset has been displayed along with the anomalies detected by the LSTM model. The red dots are anomalies. There are some scattered red dots indicating that they can be the false positives as there is no sequence of anomalies rather some isolated ones.

**A graph with a red line

Description automatically generated**

Figure 28: Detected anomalies in test samples using LSTM.

## Isolation Forest

Isolation Forest performed well in test validation part. It was able to detect the TN data points to a higher number. The best part is it didn’t detect any False Positive anomaly point. The number of False Negative was also not that much high. It was able to detect 460 actual anomalous points.

In Figure 29, the graph portrays the result of the confusion matrix. From the graph, it can be confirmed that all the anomalies detected by IF model created a sequence which marks actual anomalous points.

**A graph with blue squares and numbers

Description automatically generated**

Figure 29: Confusion matrix of trained isolation forest model.

A graph showing a red and blue line

Description automatically generated

Figure 30: Detected anomalies in test samples using Isolation Forest.

## LOF

The confusion matrix summarizes the performance of the LOF anomaly detection model. It shows four values: true positives (486), true negatives (3930), false positives (42) and false negatives (81). This analysis sheds more light on the aspect of the model which concerns the detection of anomalies and normal data points. The instances of true positive and negative ensure quite exact predictions although some few of them must be misclassified.

**A blue and white graph

Description automatically generated**

Figure 31: Local Outlier Factor confusion matrix.

From the anomaly detection graph in Figure 32 of test samples, it can be easily mentioned that there are some isolated anomalies that is detected by LOF which are misclassified by the model.

A graph showing the temperature of a person

Description automatically generated

Figure 32: Detected anomalies in test dataset using LOF.

## One class SVM

The performance of One-class SVM is less efficient in comparison with IF. It detected 246 data points as False positives. Too many false positives create unnecessary false alarms which indicated that the model is too sensitive. The number of True negative was significantly lower than IF. It was also able to more True positives than IF which is a good sign.

A blue and white chart

Description automatically generated

Figure 33: Confusion matrix of One-class SVM.

A graph showing the temperature of a person

Description automatically generated

Figure 34: Detected anomalies in test sample dataset using One-class SVM.

## K-means clustering

The confusion matrix of k-means clustering model also shows great potential. The model was able to detect 475 instances of true positives and 3966 instances of true negatives. The false positive rate was also very small – detected only 6 instances of false positive. The number of false negatives is 92 which is lower than IF model but higher than LOF and One-class SVM. Overall, the model performed really well.

A blue and white graph

Description automatically generated

Figure 35: Confusion matrix of K-means clustering.

A graph showing the temperature of a person

Description automatically generated with medium confidence

Figure 36: Anomaly detection in test sample using k-means clustering.

## Comparison among machine learning algorithms

In the present evaluation of models for detecting anomalies, five different methods - Isolation Forest, Local Outlier Factor (LOF), One-Class SVM, K-Means Clustering and LSTM are assessed according to the precision, recall and F1 metrics.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model Name** | **Precision** | **Recall** | **F1 Score** |
| Isolation Forest | 1.0 | 0.81 | 0.90 |
| LOF | 0.92 | 0.85 | 0.88 |
| One class SVM | 0.69 | 0.96 | 0.80 |
| K-means clustering | 0.99 | 0.84 | 0.91 |
| LSTM | 0.83 | 0.66 | 0.74 |

Table 4: Comparison among the model results.

The position of particular merit in that model referred to the Isolation Forest is for perfect one-to-one precision (1.0). Such indicates that all the flagged by the model anomalies are indeed anomalies. The model also demonstrates a healthy recall at 0.81, which implies that 81% of all actual anomalies are detected by the model. The F1 score which gives credit to both precision metrics is also very high at 0.90 referring to the best model.

Local Outlier Factor (LOF) also performs well and has a precision of 0.92 and a recall of 0.85 which results in an F1 of 0.88. This model shows great promise in its ability to find anomalies with less false positive results, while also reverting a large percentage of the true anomalies. It does not reach perfect precision like the Isolation Forest, yet it remains quite adequately and fairly executed.

One-Class SVM is however more recall oriented with a score of 0.96, thus almost all of the true anomalies are recognized, however, the precision is fairly low at 0.69. This means that the model does trap most of the anomalies but equally a higher number of false positives as well. This Figure of merit F1 of 0.80 indicates the extent of disbalance between precision and recall in this case making the model more skewed than the other two models viz. isolation forest and LOF.

K-Means clustering yields a very good precision of 0.99, meaning it does not classify instances which are non-anomalies as anomalies at any time. The recall score for K-means clustering is also quite high at 0.84, which shows that it is able to identify majority of actual anomalies in the dataset. The F1 Score is excellent at 0.91 which balances precision and recall. This indicates that K-means clustering not only identifies anomalies reliably but also maintains a high rate of accuracy in its predictions.

LSTM has achieved moderate performance with a precision of 0.83 and a recall of 0.66 resulting in an F1 score of 0.74. While LSTM does provide a fairly good precision level, it lags behind in recall which shows it is likely to miss more anomalies than other models in the comparison. There is room for performance improvement especially in increasing the number of true anomalies captured.

The final results of this study are based on the above data. This analysis suggests that models like Isolation Forest and K-means are preferable for tasks demanding high accuracy in anomaly detection. One Class SVM might be suitable for scenarios where capturing as many positives as possible is more critical than avoiding false positives.

## Grafana visualization

The Grafana dashboard provides a time series visualization of data, highlights anomalies and summarizes the total number of data points and anomalies. Isolation Forest model has been used for this visualization and seems to be identifying potentially abnormal behavior in the time series.

**A screenshot of a computer

Description automatically generated**

Figure 37: The dashboard of Grafana after data ingestion.

In Figure 37, the green line represents the temperature readings from sensors over time from 02/05 to 02/19. The Y-axis measures the value of this metric whereas the X-axis represents the date range. The orange dots represent anomalies detected in the data. Each dot marks a point where the anomaly detection model flagged something unusual or unexpected in the metric.

There are two gauges – blue one represents total number of entries have been used for this simulation and red one shows the number of anomalies within these total entries. From the figure it can be summarized that among 4539 entries 605 entries are detected as anomaly by IF model.

1. Conclusion

This thesis aimed to address the key challenges and methodologies associated with real-time anomaly detection in time-series data within data-intensive industrial settings. The research focused on three main questions: how to effectively perform real-time anomaly detection using time-series databases, the best practices for data preprocessing and architectural design, and identifying the most effective machine learning algorithms for industrial anomaly detection.

For the first question, it is found that real-time anomaly detection is achievable by using time-series databases along with event-driven systems. Tools like Kafka help handle fast data streams, while time-series databases, like TimeScaleDB, make it easier to store and retrieve large amounts of data quickly. Together, these technologies allow for responsive and scalable anomaly detection, which is crucial for industrial use.

In looking at best practices, it is discovered that effective detection relies on good data preparation - normalizing data, reducing noise and dividing data into meaningful segments. A system built with a microservices design and flexible data pipelines is ideal. Tt handles large data loads and supports fast, continuous monitoring. Adding data storage techniques like partitioning and sharding also helps keep the system fast and stable, even under heavy use. TimeScaleDB database already has these in-built features which makes the system robust.

Lastly, the machine learning algorithms such as isolation forests, Local Outlier Factor perform well for finding anomalies in time-series data. Each has strengths: isolation forests are effective for spotting unusual data points in large datasets. The best algorithm depends on the specific needs, like speed, accuracy and ability to adapt to changing data. The algorithm performance may also vary based on the dataset.

## Future work

The work done in this thesis can be further improved by taking some measures. Enhancing the features will improve the performance and reliability of the application.

Advanced machine learning models such as deep learning models can be trained in the future to improve the accuracy of the anomaly detection. There are more online algorithms available now-a-days which can also be used instead of offline algorithms to continuously adapt to the changes in the data. Developing models that adapt based on new patterns in data streams could enable anomaly detection to remain relevant over time without manual retraining.

Even though analysis of scalable system for data-intensive scenarios has been discussed in this thesis, the dataset used for implementation was not large enough. So, the implementation of the application was done with monolithic architecture. In future, a more data-intensive dataset can be used to test the performance of the application. Microservice architecture can also be implemented to handle large-scale data which is more scalable than monolithic architecture.

# References

|  |  |
| --- | --- |
| [1] | S. Sinha, "IOT analytics," 3 Sep. 2024. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/. |
| [2] | J. Azar, A. Makhoul, R. Couturier and J. Demerjian, "Robust IoT time series classification with data compression and deep learning," *Neurocomputing ,* vol. 398, pp. 222-234, 2020. |
| [3] | Y. Zhu, M. Imamura, D. Nikovski and E. Keogh, "Introducing time series chains: a new primitive for time series data mining," *Knowledge and Information Systems ,* vol. 60, pp. 1135-1161, 2019. |
| [4] | L. Tianyuan, L. Wang and X. Zhao, "Review of anomaly detection algorithms for data streams," *Applied Sciences,* vol. 13, no. 10, p. 6353, 2023. |
| [5] | V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR),* vol. 41, no. 3, pp. 1-58, 2009. |
| [6] | K. Shaukat, T. M. Alam, S. Luo, S. Shabbir, I. A. Hameed, J. Li, S. K. Abbas and U. Javed, "A review of time-series anomaly detection techniques: A step to future perspectives," in *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 1*, (2021): 865-877. |
| [7] | M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," in *arXiv preprint arXiv:2004.00433*, 2020. |
| [8] | M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain and A. J. Aljaaf, "A systematic review on supervised and unsupervised machine learning algorithms for data science," in *Supervised and unsupervised learning for data science*, (2020): 3-21. |
| [9] | X. Zhu, "Semi-supervised learning literature survey.," in *Computer Sciences TR*, 1530 (2008): 60-60. |
| [10] | A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," in *Computer Networks, Volume 51, Issue 12*, 2007. |
| [11] | S. Schmidl, P. Wenig and T. Papenbrock, "Anomaly detection in time series: a comprehensive evaluation," in *Proceedings of the VLDB Endowment 15, no. 9*, (2022): 1779-1797. |
| [12] | O. Kramer, "K-nearest neighbors," in *Dimensionality reduction with unsupervised nearest neighbors*, 2013, pp. 13-23. |
| [13] | D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," in *Data & knowledge engineering 60, no. 1*, (2007): 208-221. |
| [14] | M. Gopinath, F. A. Rahim and Z.-A. b. Ibrahim, "A survey on anomalies detection techniques and measurement methods," in *2018 IEEE conference on application, information and network security (AINS)*, (2018): 81-86. |
| [15] | N. Sarfaraz, M. Leghari, M. A. Rajput, S. S. Zia and J. Shabir, "A Systematic Review of Anomaly Detection Using Machine and Deep Learning Techniques," in *Quaid-e-Awam University Research Journal of Engineering Science and Technology 20, no. 1*, (2022): 83-94. |
| [16] | S. Mahsa, C. Leckie, J. C. Bezdek, T. Vaithianathan and X. Zhang, "Fast Memory Efficient Local Outlier Detection in Data Streams," in *IEEE Transactions on Knowledge and Data Engineering 28*, no. 12 (2016): 3246-3260. |
| [17] | Y. Xinying, S.-Y. Liong and V. Babovic, "EC-SVM approach for real-time hydrologic forecasting," in *Journal of Hydroinformatics 6*, no. 3 (2004): 209-223. |
| [18] | K. Maria, A. Gounaris, A. N. Papadopoulos, K. Tsichlas and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *2011 IEEE 27th International Conference on Data Engineering*, (2011): 135-146. |
| [19] | L. Yuh-Jye, Y. Yi-Ren and W. Y.-C. Frank, "Anomaly detection via online oversampling principal component analysis no. 7," in *IEEE transactions on knowledge and data engineering 25*, (2012): 1460-1470. |
| [20] | D. Novoa-Paradela, O. Fontenla-Romero and B. Guijarro-Berdiñas, "Online learning for anomaly detection via subdivisible convex hulls," in *2020 International Joint Conference on Neural Networks (IJCNN)*, (2020): 1-8. |
| [21] | D. Sharmishta, S. Roy, B. Patel, S. Purandare and M. Kucheria, "Very Fast Decision Tree (VFDT) algorithm on Hadoop," in *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, (2016) : 1-7. |
| [22] | M. Chaitanya, G. I. Webb and M. Salehi, "Extremely Fast Decision Tree," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (2018): 1953-1962. |
| [23] | Erhan, Laura, M. Ndubuaku, M. D. Mauro, W. Song, M. Chen, G. Fortino, O. Bagdasar and A. Liotta., "Smart anomaly detection in sensor systems: A multi-perspective review," in *Information Fusion*, 67 (2021): 64-79. |
| [24] | Habeeb, A. R. Ahamed, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed and M. Imran, "Real-time big data processing for anomaly detection: A survey," 45 (2019): 289-307. |
| [25] | D. Y. Deshpande and S. R. Rahman, "Edge-Based Real-Time Sensor Data Processing for Anomaly Detection in Industrial IoT Applications," in *Research Journal of Computer Systems and Engineering 4, no. 2*, (2023): 16-30. |
| [26] | Y. Dit-Yan and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," in *Pattern recognition 36*, no. 1 (2003): 229-243. |
| [27] | S. Ahmad, A. Lavin, S. Purdy and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," in *Neurocomputing 262*, (2017): 134-147. |
| [28] | M. Teng, "Anomaly detection on time series," in *2010 IEEE International Conference on Progress in Informatics and Computing*, vol. 1, pp. 603-608. IEEE, 2010. |
| [29] | C. Bulla and M. N. Birje, "Anomaly Detection in Industrial IoT Applications Using Deep Learning Ap-proach," in *Artificial Intelligence in Industrial Applications: Approaches to Solve the Intrinsic Industrial Optimization Problems*, (2022): 127-147. |
| [30] | P. Yan, A. Abdulkadir, P.-P. Luley, M. Rosenthal, G. A. Schatte, Benjamin F. Grewe and T. Stadelmann, "A Comprehensive Survey of Deep Transfer Learning for Anomaly Detection in In-dustrial Time Series: Methods, Applications, and Directions," in *IEEE Access*, (2024). |
| [31] | T. Tziolas, K. Papageorgiou, T. Theodosiou, E. Papageorgiou, T. Mastos and A. Papadopoulos, "Autoencoders for Anomaly Detection in an Industrial Multivariate Time Series Da-taset," in *Engineering Proceedings 18*, no. 1 (2022): 23. |
| [32] | Y. Yong, X. Si, C. Hu and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," in *Neural computation 31*, no. 7 (2019): 1235-1270. |
| [33] | L. F. Tony, K. M. Ting and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*, (2008): 413-422. |
| [34] | A. Omar, R. Alsini, T. Soule and X. Ma, "A review of local outlier factor algorithms for outlier detection in big data streams," in *Big Data and Cognitive Computing 5*, no. 1 (2020): 1. |
| [35] | A. Mennatallah, M. Goldstein and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, (2013) : 8-15. |
| [36] | "Github repositiry - numenta/NAB," [Online]. Available: https://github.com/numenta/NAB/tree/master/data/realKnownCause. |
| [37] | C. Hansi, H. Ma, X. Chu and D. Xue, "Anomaly detection and critical attributes identification for products with multiple operating conditions based on isolation forest," in *Advanced Engineering Informatics*, 46 (2020): 101139. |
| [38] | "LOF Image," [Online]. Available: https://scikit-learn.org/stable/auto\_examples/neighbors/plot\_lof\_outlier\_detection.html. |
| [39] | "TimeScale," [Online]. Available: https://www.timescale.com/. |
| [40] | "Grafana," [Online]. Available: https://grafana.com/. |
| [41] | "Docker," [Online]. Available: https://www.docker.com/. |
| [42] | "Building a scalable database using TimeScale," [Online]. Available: https://www.timescale.com/learn/building-a-scalable-database. |