CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Study on Isolation Forest for Anomaly Detection in Cloud-Based Systems

Master's Thesis in Computer Science and Engineering

FREDRIK NÄTTERDAL

KARL OLAUSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

A Study on Isolation Forest for Anomaly Detection in Cloud-Based Systems

FREDRIK NÄTTERDAL
KARL OLAUSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

A Study on Isolation Forest for Anomaly Detection in Cloud-Based Systems
FREDRIK NÄTTERDAL
KARL OLAUSSON

Supervisor: Ricardo Caldas, Department
Advisors: Simon Evaldsson & Adam Davidsson, WirelessCar
Examiner: Daniel Strüber, Department

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

A Study on Isolation Forest for Anomaly Detection in Cloud-Based Systems
FREDRIK NÄTTERDAL
KARL OLAUSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The need for effective monitoring solutions increases as more organizations migrate to a cloud infrastructure. The leading cloud providers offer their own monitoring services, but they lack customizability and are black-box, making it hard to understand and control their behavior. This thesis investigates developing and deploying a custom monitoring service to the cloud. This will be done by applying the Isolation Forest (iForest) algorithm as an anomaly detection tool for monitoring cloud services within Amazon Web Services (AWS). While iForest is a powerful unsupervised learning algorithm, it is made for static analysis. Applying it to streaming data introduces some challenges, such as concept drift and seasonal changes in the data. Our research addresses these challenges by tailoring iForest to cloud monitoring.

As many of you are aware, more and more companies have been migrating their operations to the cloud in order to enhance flexibility, scalability, and efficiency in their operations. In this thesis, we studied iForest for detecting anomalies in CloudWatch metrics data in the context of WirelessCar. The developed model demonstrated superior performance compared to state-of-the-art alternatives. Additionally, we deployed the model to the cloud through AWS where it was used to detect anomalies in live data from a data stream. We have summarized the findings and provided practical guidelines for anomaly detection development and cloud deployment. These guidelines aim to assist practitioners and researchers with integrating anomaly detection for cloud monitoring.

Keywords: software, engineering, anomaly, detection, isolation, forest, streaming, guidelines, AWS.

Acknowledgements

We would like to express our gratitude to the people involved in the project, who have supported us throughout the journey with their knowledge and experience. To begin with, we would like to thank our academic supervisor, Ricardo Caldas, who has provided guidance on how to approach the thesis and helped us overcome any roadblocks along the way. We would also like to thank our advisors at WirelessCar, Simon Evaldsson, and Adam Davidsson, who have shared their knowledge and helped us integrate our system into the workflow of WirelessCar. Lastly, we would like to acknowledge Sami Fatmi at WirelessCar, who took an interest in our thesis early on and has since helped us with numerous things, like answering our many machine learning-related questions.

Fredrik Nätterdal, Gothenburg, June 2024
Karl Olausson, Gothenburg, June 2024

Contents

List of Figures

List of Tables

1
Introduction

As most modern software organizations have migrated from traditional infrastructures to cloud infrastructures in order to facilitate the increasing data processing demands, monitoring of cloud services has become increasingly important. Amazon Web Services (AWS) monitors these services through Amazon CloudWatch, enabling organizations to retrieve metrics for their AWS applications and resources. These metrics reflect a system's operational health and thus must be analyzed for potential deviations from what is considered normal behavior. Such deviations are known as anomalies.

Anomaly detection is the process in which data is analyzed to find deviations. This process is usually automated using machine learning (ML), and several techniques have previously been used for the task [29]. Among these techniques, the Isolation Forest (iForest) algorithm has become a well-established method for anomaly detection as it utilizes unsupervised learning and has a linear time complexity, making it excellent for processing large amounts of data. iForest is a tree-based algorithm that leverages the concept of isolation to detect anomalies [26]. While iForest is a powerful tool for unsupervised anomaly detection, it has some limitations. Firstly, it requires prior knowledge about the anomaly rate in data to make accurate predictions. Secondly, it is limited because it is made for static data analysis. Applying it in data streams requires adapting to changes in data distribution, known as concept drift, and accounting for seasonal fluctuations. Research on addressing limitations with the iForest algorithm [16][19][22] and concept drift [17][30][32][33] have been published, but, to our knowledge, iForest has not previously been applied to monitor cloud services.

As the demand for monitoring of cloud services increases, so does the need for robust tools designed for detecting anomalies in those services. AWS offers its own anomaly detection tool as one of its monitoring services, but the specifications of the tool are not publicly available, i.e., it is a black-box piece of software. Thus, it does not offer the possibility of understanding its decisions and customizing the tool for a specific purpose. Furthermore, no other built-in tools are available for anomaly detection in the monitoring service.

To address this problem, we conducted a case study in collaboration with WirelessCar, which centers on applying iForest in a data streaming context. Thus, the case covers developing and deploying an anomaly detection model based on the iForest algorithm applied to metrics data from a monitoring service. The model

includes a combination of different components to tailor it to the nature of the data. Before deploying the model, it was evaluated against five state-of-the-art anomaly detection models in data streams representing the baseline. The result indicates that the developed model has great potential as it outperforms the baseline when applied to a dataset originating from Amazon CloudWatch. The model was then deployed to AWS as an alternative to CloudWatch's built-in anomaly detection model.

In this thesis, we explore different state-of-the-art extensions to iForest to gain a foundational understanding of the algorithm and how it can be improved. A model is developed based on the insights gained and the characteristics of the data. The anomaly detection model is then deployed using SageMaker, an Amazon service used to build, train, and deploy ML models. Finally, a set of guidelines is provided based on the knowledge accumulated throughout the development phase. These guidelines serve as recommendations for practitioners working with AWS and researchers studying anomaly detection in cloud systems. This project is done in collaboration with WirelessCar, a company specializing in software solutions for connected vehicles, which granted us access to data streams in CloudWatch to use for anomaly detection.

1.1 Problem Statement and Purpose of the Study

With an increasing number of companies migrating their software to cloud infrastructures comes a need to properly monitor cloud services. As part of its CloudWatch service, AWS provides a built-in anomaly detector for streaming data, which operates as a black-box model. The implication following this is that the model is limited in its configurability and the details of how it operates. This lack of flexibility prevents companies from tuning the anomaly detection model to align with their specific needs. As a result, potential issues can emerge, such as certain anomalies being undetected or false alarms being raised.

This study aims to investigate the possibilities of improving the current monitoring capabilities of Amazon CloudWatch. This will be done by exploring extensions of the iForest algorithm applied to streaming data and refining a state-of-the-art model for CloudWatch metrics data. The model will then be used on real data from CloudWatch and evaluated against other state-of-the-art anomaly detection algorithms. By proposing an enhanced approach to monitoring cloud-based systems, this thesis has the potential to contribute to both the fields of software engineering and cloud computing, offering value to both academia and the industry.

1.2 Research Questions

This thesis aims to enhance iForest and improve its anomaly detection capabilities in data streams in a cloud computing environment. This research is centered around understanding the state-of-the-art of iForest for streaming data and the challenges of deploying it as a cloud service. The research questions below have been formu-

lated to build foundational knowledge about iForest, further propose improvements to an existing algorithm, and contribute to advancements in research and software engineering.

RQ1 - Current state of anomaly detection in streaming data using iForest: *What are the main challenges with iForest in streaming data, and what state-of-the-art models based on iForest for data streams are currently available?*

This question aims to understand current research on improvements to the iForest algorithm. By studying these extensions, the goal is to find their respective strengths and weaknesses and, using these insights, identify a suitable model to improve further.

RQ2 - Optimizing iForest for CloudWatch metrics: *How can an extension of iForest be applied effectively in the specific context of metrics data from a cloud service provider?*

By utilizing the insights gained from RQ1, the goal of this research question is to determine how an anomaly detection model based on iForest can be applied to perform optimally in the case of CloudWatch metrics data. This will involve developing a composite model based on an extension of the iForest algorithm to handle the specific characteristics of the data coming from CloudWatch.

RQ3 - Guidelines for integrating anomaly detection in the industry: *How can the findings from the study provide guidelines that will aid software engineering practitioners in the process of deploying and integrating anomaly detection into a cloud-based workflow?*

The purpose of this question is to establish a set of guidelines for deploying an anomaly detection model as a cloud service, specifically in AWS. The guidelines will be based on the theoretical and practical findings gained from RQ1 and RQ2 and throughout the project. We aim to aid industry practitioners by sharing these insights, as they can serve as a foundational framework for developing and deploying anomaly detection systems within a cloud-based infrastructure, making integrating models better suited for their specific needs easier.

1.3 Study Design

This thesis will adopt a Design Science Research (DSR) methodology, as it offers a systematic approach to both creating practical solutions, such as developing software, and contributing to academic knowledge with relevant research questions [21]. Following the DSR guidelines, the thesis will be divided into three cycles. Each cycle will include the stages *awareness*, *solution*, and *validation*. Awareness refers to the activities conducted to acquire the necessary foundation to answer relevant research questions. Solution specifies the outcome or artifact produced during a cycle. Validation describes the way the artifact has been evaluated.

One research question will be of main concern for each cycle, but as recommended, all questions should be worked on to some degree. The first cycle will be related to **RQ1**, and thus will be where most of the literature studies are conducted. The cycle's outcome will be a deep understanding of current state-of-the-art anomaly detection techniques related to iForest in data streams. The second cycle will be centered around **RQ2**. Based on the insights gained from the first cycle, this phase will include improving the chosen anomaly detection model's capabilities in a cloud monitoring service by combining it with other techniques and algorithms. Finally, in the last cycle, **RQ3** will be answered by synthesizing guidelines relating to developing the final model, deploying it to AWS, and using it to detect anomalies in data streams.

1.4 Significance of the Study

This study aims to provide valuable contributions to the software engineering industry and research. The contributions to the software engineering industry will be guidelines that can aid software engineers with integrating anomaly detection into cloud-based systems. By improving anomaly detection in cloud-based systems and ensuring the high demands of modern software systems are met, contributions from this study can extend to the broader field of quality assurance in software engineering. Additionally, we aim to extend the current body of research on anomaly detection in cloud computing by proposing a model that effectively detects anomalies in live metrics data. The findings from this study can provide a foundation for future research in the field.

1.5 Limitations

The scope of this thesis will mainly include the improvement of an extension of the iForest algorithm as well as the integration of it into the monitoring workflow of AWS. Many interesting methods have been proposed in the literature for anomaly detection, but due to time constraints, the scope has been focused on only iForest extensions for streaming data. Additionally, evaluating all relevant anomaly detection models will not be feasible as only a few of their repositories are public, and there is not enough time to re-implement them all.

2
Background and Related Work

This chapter presents an overview of anomaly detection, with emphasis on the iForest algorithm and frameworks used in the project. Extensions to the iForest algorithm are also described.

2.1 Anomaly detection

Anomaly detection refers to identifying observations or patterns in data that deviate from normal behavior [11]. In this task, data is typically divided into two categories: normal data, an observation that conforms to a predefined expected pattern or distribution, and anomalous data, which significantly deviates from this expected behavior. Although identifying anomalies might seem trivial, given their concept, it is far more complex in reality. Anomalies vary greatly depending on the context, which results in the performance of anomaly detection methods being highly dependent on their application domain.

Anomaly detection has been applied in various domains. For example, in healthcare monitoring, anomaly detection can be used to identify unexpected features in a scan, potentially suggesting a severe medical condition. Similarly, anomaly detection can be used to identify potential security breaches in cybersecurity. Given the wide range of application domains and how critical of a task it can be, there has been a lot of interest in researching the topic, leading to many different anomaly detection methods being developed [11].

2.1.1 Anomalies in Time-Series Data

In time-series data, mainly three types of anomalies occur [11]. They vary in characteristics, causing anomaly detection models to perform differently for different types. Therefore, understanding the distinction between them is crucial when developing an anomaly detection algorithm.

Point Anomaly

The point anomaly is the least complex and is defined as a single data point significantly deviating from the general data. In a visualization of a time-series dataset, it can be seen as a sudden spike or dip in the trace.

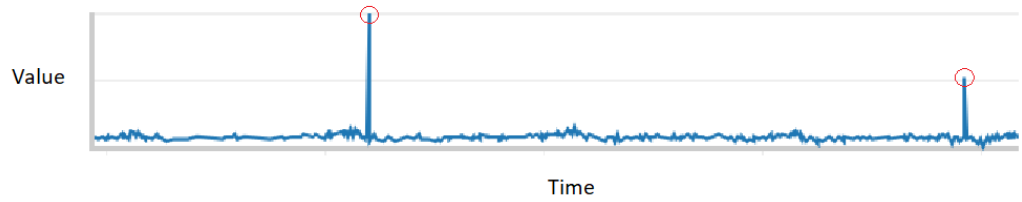

Figure 2.1: Example of point anomalies in time-series data where the anomalies are circled in red.

Contextual Anomaly

Contextual anomalies are similar to point anomalies in deviating from the otherwise smooth trace of data points. The main distinction is that a contextual anomaly is evaluated with the data's seasonal changes. For instance, the number of incoming requests to a transportation service will be highest in the morning and afternoon when more users drive. An anomaly in that context would manifest itself as a local outlier in the trend of the incoming data. The value might not be unusual to see at other times of the day and is thus not detected as a point anomaly, but it is anomalous regarding the context.

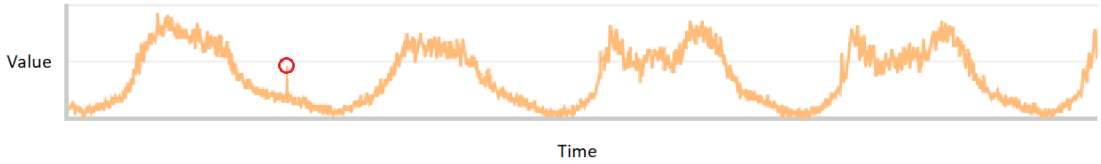

Figure 2.2: Visualization of a contextual anomaly, circled in red. The point is within the range of normal values, but in its context, it is deviating from the usual behavior.

Collective Anomaly

The most complex type of anomaly is the collective anomaly. It differs from the previously mentioned types as it is identified by analyzing multiple data points. A collective anomaly consists of data points that appear normal individually but together form an anomaly. It is generally applicable in scenarios where a specific sequence of data points can indicate system errors or malicious activity.

2.1.2 Anomaly detection learning methods

In anomaly detection, data is usually referred to as labeled or unlabeled. Labeled data implies that each dataset observation has a label indicating whether it can be considered normal or abnormal, i.e., if the data is an anomaly or not. On the other hand, the observation in an unlabeled dataset does not have a label indicating its nature. The nature of a dataset needs to be considered when deciding which anomaly

detection technique to use. The importance of distinguishing labeled and unlabeled data stems from anomaly detection techniques being designed for a specific learning method, categorized as supervised, semi-supervised, or unsupervised learning [11].

Supervised learning is the preferred method for training ML models when labeled data is available. The premise of supervised learning for anomaly detection is based on each observation label being correctly classified as normal or anomalous and that it covers the entire spectrum of anomalies. If this is the case, anomaly detection algorithms that rely on supervised learning can typically produce predictions with very high accuracy. While supervised learning can produce good results under the right circumstances, it is limited by the need for labeled data, which is not feasible in some cases, such as for anomaly detection in data streams.

Unsupervised learning, unlike supervised learning, does not require the input data to have predefined labels. Instead, unsupervised learning captures the underlying patterns and characteristics of the data, enabling models to make predictions based on deviation from these patterns. In terms of anomaly detection, unsupervised learning algorithms offer a variety of advantages. Firstly, the models created from these algorithms are highly adaptable, meaning that they can be applied to new data without the task of labeling. Secondly, unsupervised learning can detect hidden patterns in the data, potentially discovering new anomalies. Lastly, unsupervised learning provides the possibility of anomaly detection for evolving data streams, meaning that these types of models can efficiently be retrained once the patterns in the data exhibit change.

Semi-supervised learning is an approach to training ML models that combines supervised and unsupervised learning by leveraging both labeled and unlabelled data. Semi-supervised learning is typically applied when there is a small amount of labeled data but a large amount of unlabelled data. For most anomaly detection models utilizing semi-supervised learning, the process only involves training on data that is normal [11].

2.1.3 Data Normalization

One of the general challenges in ML is imbalanced distributions of the data. Training models on such data can result in inaccurate predictions as different data classes are not represented equally [34]. This issue can be addressed through data normalization, which refers to modifying data distributions by scaling and handling skewness in data [14]. Scaling is done by changing the range and scale of different features to be more similar in their means and standard deviation. This is an important step in algorithms like K-Nearest-Neighbor, where distances between data points greatly affect predictions. On the other hand, handling skewness is done to transform the distribution to be more Gaussian, which is crucial for some algorithms.
A technique that can be used for data normalization, specifically to handle skewness, is a power transformation of the data. Applying a power transform to data causes the data to be approximated as a more normal distribution and stabilizes

the variance. It has previously been applied in anomaly detection as a preprocessing tool, with the main purpose of narrowing the value range of anomalies in data [31]. In the paper, the transformation of the data was shown to enhance the model's performance in identifying anomalies. This was due to the power transformation addressing the issue of some anomalies being magnitudes larger than other anomalies, making the anomalies closer to the normal baseline go undetected. Transforming the data was shown in the paper to enhance the model's ability to identify more anomalies.

In the family of power transformers, two widely adopted algorithms are the Box-Cox transform and the Yeo-Johnson transform, both available in the Scikit-Learn library [3]. The definition of the Box-Cox transform is given by [10]:

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(y) & \text{if } \lambda = 0, \end{cases}$$

Here, y is the input data, and λ is the transform parameter, which is estimated with maximum likelihood to optimize stabilization of variance and minimization of skewness. An inherent limitation of the transform is that it can only be used with positive input values. An extension was proposed to the transform called the Yeo-Johnson transform [36]. It is built upon the Box-Cox algorithm and allows the input data to be negative.

2.1.4 Evaluation Metrics

Evaluating anomaly detection models requires a slightly different approach than other ML models. A common performance metric is accuracy, which represents the percentage of correct predictions done by a model. However, with anomaly detection, the characteristics of the data can cause the metric to show an incorrect reflection of the model performance. This is due to the significantly lower presence of anomalies than normal data in many datasets. If a model can make accurate predictions about normal points but is less accurate in predicting anomalies, intuitively, it should impact the performance metric. However, using accuracy in this scenario will indicate that the model has high predictive performance due to the skewness in the data when, in reality, it is only fit to classify normal data points.

Different metrics have been used to evaluate anomaly detection algorithms, and commonly applied metrics are True Positive Rate (TPR), False Positive Rate (FPR), F1-score, and AUC-ROC [27]. F1-score is an evaluation metric designed to work the same way independently of distribution imbalances in a dataset. It uses the metrics Precision and Recall to calculate a performance percentage, which is based on True Positives (TP), False Positives (FP), and False Negatives (FN). Below are the algorithms for calculating F1-score:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Another commonly used metric is the Area Under the Curve of the Receiver Operating Characteristic (AUC-ROC, or AUC). The ROC curve demonstrates how the FPR and TPR measures are contrasted graphically. The values are represented in opposing axes, and the curve between them reflects how well the model performs for different thresholds. AUC translates the curve to a performance metric by calculating the area under the curve. Similar to the F1 score, AUC excels in scenarios where the data distribution is imbalanced.

2.1.5 Anomaly Detection in Streaming Data

Anomaly detection is generally done on static datasets or batches of data sent to the model. The third way anomaly detection is used is with streaming data which is when there is a constant incoming flow of live data. Streaming data analysis comes with challenges that require unique approaches compared to anomaly detection on static data [33]. Some of the challenges are:

Real-time processing: The application of anomaly detection in streaming data comes from the need to identify deviations in incoming data as soon as it is received. This puts the requirement on the model to quickly analyze streamed data points.

Changes in data distribution over time: A phenomenon introduced in data streams is that trends tend to change over time and is known as concept drift. As data distributions drift from what models have been trained on, they become outdated and unfit to handle newer data.

Infinite nature of data streams: As live data essentially have an endless stream of incoming data, they are infinite in their nature. This comes with high memory requirements, and storing all historical data in most systems is not feasible.

Strategies have been proposed to counter the inherent challenges of streaming data, usually through the use of sliding windows and drift detection algorithms such as KSWIN and ADWIN [17][30][32][33].

Sliding Window

The sliding window technique is commonly used for storing batches of data from a data stream. For instance, it has been proposed as an extension to iForest to enable anomaly detection on live data [13]. A sliding window is essentially a list of data points with a predefined length `window_size`. As new data arrives it is placed into

the window until it has been filled. Typically, the model keeps track of the recently filled window `prec_window` together with the active window `window`. As `window` reaches its maximum length `window_size`, it replaces `prec_window` and is then cleared. Using this approach, the anomaly detection model requires significantly less memory as it only stores relevant data. Additionally, `prec_window` plays a crucial role when concept drift is detected in the data, where it is used to retrain the model, as the data distribution has changed and the model needs to be trained on recent data. An example of concept drift is illustrated in Figure 2.3

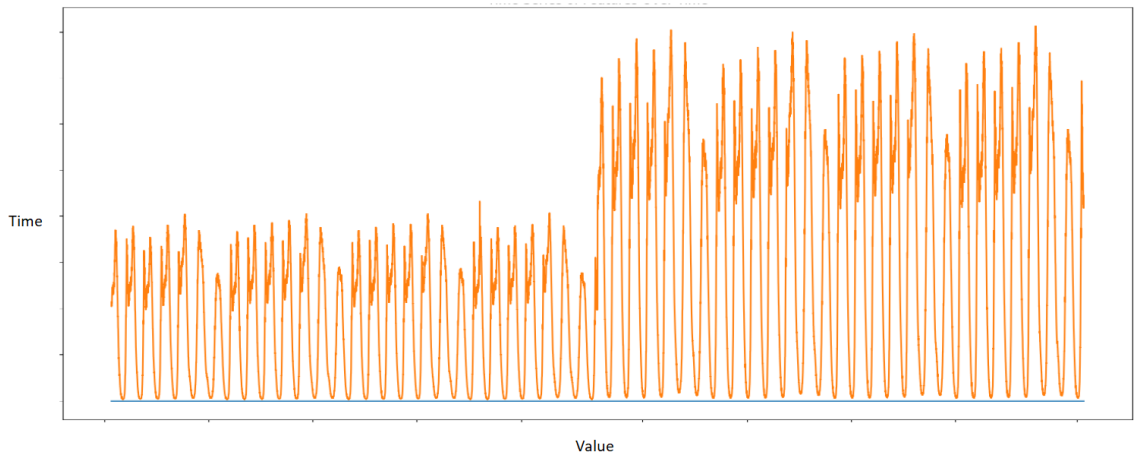

Figure 2.3: Example of how concept drift can look in time-series data.

Drift Detection

In earlier extensions to iForest for streaming data, such as the model proposed by Ding & Fei [13], concept drift is detected in a relatively primitive way. The model has a constant `u`, which is the anomaly threshold. After the anomaly detection is completed on a window, the rate of anomalies within the window is calculated. The model is retrained if the rate is above the threshold `u`. One of the main limitations of using this approach is that it requires previous knowledge of the anomaly rate in the data to set the threshold correctly. Several more complex drift detection techniques have later been proposed, like ADaptive WINdowing (ADWIN) [9] and Kolmogorov-Smirnov WINdowing (KSWIN) [28].

ADWIN: ADWIN maintains an adaptive sliding window of the most recent data points in a data stream. In this context, adaptive refers to the dynamic adjustments of the sliding window length that can be done during processing. To make it possible for the algorithm to detect change, it divides the sliding window, W, into two sub-windows, denoted as *W1* and *W2*. *W1* corresponds to the older entries in W, while *W2* corresponds to the more recent ones. The algorithm continuously compares the average of the values in *W1* and *W2* to determine if the distribution differs more than a predefined threshold, δ; if it does, a drift has occurred.

KSWIN: KSWIN is based on the Kolmogorov-Smirnov (KS) test, a non-parametric

statistical test used for identifying significant differences between two distributions. The KSWIN model keeps track of a sliding window of a constant size. As a new data point arrives, it is added to the window, while the oldest entry is discarded. The window is split into two parts, from which two distributions are created. Distribution R represents the r most recent entries, while W is created by uniformly drawing samples from the split containing older data points. The two distributions are assumed to represent old and new concepts in the data, respectively. After applying the KS test to these distributions, drift can be detected using the following formula:

$$\text{dist}(R, W) > \sqrt{\frac{-\ln(\alpha)}{r}}$$

Here, `dist`(R, W) relates to the distance of the empirical cumulative data distribution calculated by the KS-test. α is the probability for the test statistic of the KS-test, which can be changed manually to adjust the sensitivity of the drift detection.

2.2 Isolation Forest

iForest is a tree-based algorithm that is specifically designed for anomaly detection [26]. It fundamentally differs from most other anomaly detection techniques as it does not rely on modeling the normal patterns in the data. Instead, iForest directly tries to isolate anomalies from the rest of the data using the premise that anomalies significantly differ from the normal data. By relying on the concept that anomalies are easily isolated, iForest can identify outliers with a linear time complexity that requires a low amount of memory, making it very effective for high dimensional data sets. The original implementation of iForest is used for static analysis, making it effective in identifying point anomalies. For the algorithm to be able to capture contextual and collective anomalies, which are generally related to time-series data, modification to how the algorithm processes the data is required.

iForest, as the name implies, constructs a forest composed of isolation trees (iTrees), with each iTree being a binary decision tree. The process of building these decision trees is quite simple. It works by randomly selecting a feature in the data and then randomly splitting this feature between its minimum and maximum value. After recursively repeating this process until each data point is isolated or until specified termination conditions have been met, the construction of an iTree is finished. This process continues until a predefined number of trees have been created and the model is initialized.

iForest identifies anomalies based on each data point's anomaly score, which essentially is its degree of isolation within the forest. This score is determined by passing each data point through all of the iTrees and calculating the average path length from the tree's root to the point of isolation. Since anomalies significantly differ from the rest of the data, they require fewer splits to be isolated on average. Thus, a shorter average path length signifies a higher likelihood of a data point

being an anomaly. The final prediction is made by comparing the anomaly scores to a predefined threshold. This threshold can be set by specifying the approximate rate of anomalies in the data.

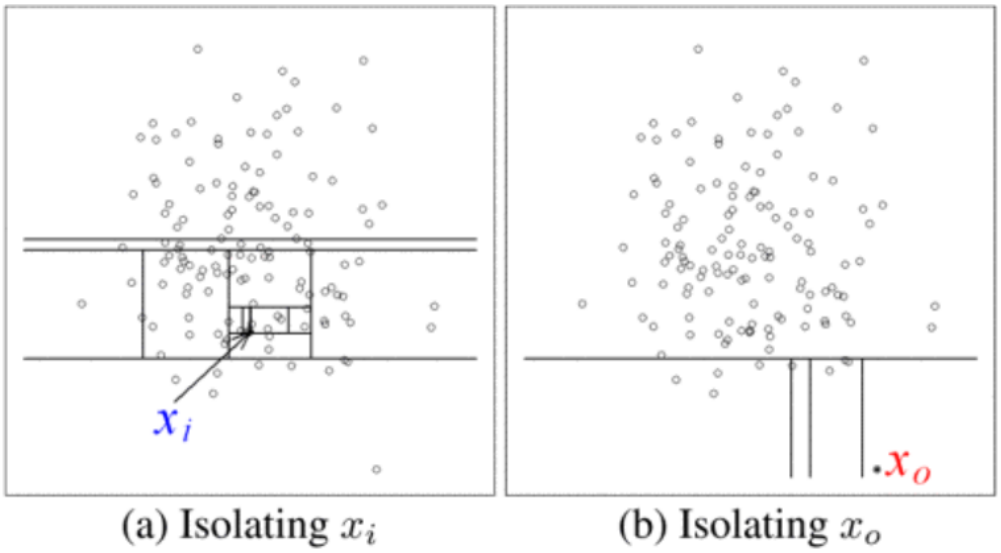

Figure 2.4: Visualization of how the splitting for a normal data point (a) and an anomalous data point (b) is done to isolate them. Isolating x_o requires fewer splits, resulting in a higher anomaly score. Image taken from the paper on iForest by Liu et al. [26].

2.2.1 Extended Isolation Forest

Since its publication in 2008 [26], iForest has been the subject of several research projects with the intent to explore the possibility of further improving the algorithm. The most widely used iForest extension is Extended Isolation Forest (EIF) [16]. It was developed to mitigate potential limitations in how the original iForest algorithm produced its anomaly scores. It was revealed when analyzing iForest that its branching procedure, i.e., the splitting procedure, produced biases in the anomaly scores. The bias appeared because the branch cuts were always parallel to the feature axes. In other words, each time the data was split, it only considered one feature, effectively ignoring the potential relationship between multiple features. Due to this type of branching, the algorithm makes many unnecessary branch cuts in sparse areas of the feature space, which produces a bias in the anomaly scores.

EIF addressed this limitation by altering the way branch cuts are made. In its predecessor, iForest, branch cuts were made by first selecting a random feature and then selecting a random value within the range of that feature. Instead, EIF selects a random slope and a random intercept on that slope. For a given dataset, which has N dimensions, the selection of the random slope is equivalent to selecting a

normal vector $\vec{n}$ uniformly over the N-sphere. For the intercept point, $\vec{p}$, the process involves drawing a value from a uniform distribution over the range of values at each branching point. When the slope and the intercept have been selected, the branching criteria for splitting a data point $\vec{x}$ is defined by the following relationship:

$$(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0$$

By introducing this change to the branching procedure, EIF allows splits to occur in any direction, as opposed to the limited axis-parallel splits, thus reducing the bias present in iForest. The new branching method also enables EIF to capture the potential relationships between multiple features, improving its performance on high-dimensional datasets. It has been demonstrated that EIF maintains the same level of computational efficiency as iForest, making it the preferred choice for applying anomaly detection. However, like iForest, EIF cannot detect anomalies in streaming data. To our knowledge, no previous research has studied its application in a streaming context despite its performance being better than iForest's.

2.3 Cloud Computing

Cloud computing can be described as the on-demand delivery of various services related to data storage, processing, and management over the Internet. Among the largest cloud service providers are Microsoft Azure [6] and Amazon Web Services (AWS) [7]. They offer flexible and scalable solutions for complex and demanding tasks through a pay-per-use model, where pricing varies depending on resource usage. In WirelessCar, AWS is the foundation for the cloud infrastructure. AWS supplies various services for a wide range of use cases [2]. In this project, several AWS services will be used to a small degree, but two have especially high relevance to the study. These services are CloudWatch and SageMaker, which will be explained in more detail.

2.3.1 CloudWatch

Amazon CloudWatch is essentially a repository that monitors and stores metrics related to services and resources within AWS [4]. The first use case of CloudWatch is to display different service metrics. For instance, metrics such as CPU utilization, network traffic in a data stream, or disk read/write success rate can be displayed. Another feature of CloudWatch is raising alarms if some unexpected behavior is seen in a metric, usually when the metric deviates from a certain threshold level for an extended time frame. When alarms are triggered, CloudWatch can initiate different actions, such as broadcasting the alarm to a notification service. Alarms can also trigger other actions, like auto scaling, which adjusts the size of a virtual machine based on CPU utilization.

In addition to monitoring various metrics, CloudWatch offers a built-in anomaly detection model, which can be accessed directly from the interface displaying the metrics. The details of the model are not disclosed; only that it operates based on

ML algorithms. It can be trained on up to two weeks of data and adapt to concept drift in incoming data streams. When configuring the model, it is possible to set a detection threshold to adjust how sensitive the model is to sudden changes in the data. Additionally, it can be set only to detect data points larger or smaller than the historical data. The anomaly detection model is an extension of CloudWatch's alarm service and can be configured to trigger an alarm if an anomaly is detected.

2.3.2 SageMaker

Amazon SageMaker is an ML service that enhances the process of building, training, and deploying ML models by making it faster and easier [5]. This is done in multiple ways, like providing a multitude of built-in models, automatic tuning of the models, and fast deployment. Once the model is deployed, SageMaker offers additional features like auto scaling, which dynamically adjusts the number of instances used for the model based on the current computational requirements. Moreover, SageMaker provides metrics for accuracy and drift to ensure the quality of the model can be maintained in real time.

As previously mentioned, SageMaker has various ML models ready for use in different cases. For anomaly detection, the algorithm *Random Cut Forest* (RCF) can be used [15]. RCF is similar to iForest in that it utilizes an ensemble of trees, and anomalies are identified based on their path lengths within these trees. However, RCF differs from iForest in how trees are constructed. While iForest makes a random selection of features and then performs splits based on these features to isolate points, RCF instead randomly partitions data by making cuts across different dimensions of the data space without a prior selection of features. This approach was proposed to solve an issue with iForest, where its application in high-dimensional datasets leads to reduced accuracy due to many dimensions being irrelevant to the anomaly detection task.

3
Methodology

This chapter presents the methodology for each of the cycles in the thesis. Cycle 1 includes the systematic literature review on iForest extensions for streaming data. Cycle 2 gives insight into the developed algorithm and its theory, followed by an evaluation and comparison to other models. Finally, cycle 3 showcases the process of integrating the final model into the workflow of WirelessCar. An overview of the intended workflow within and between the cycles is illustrated in Figure 3.1

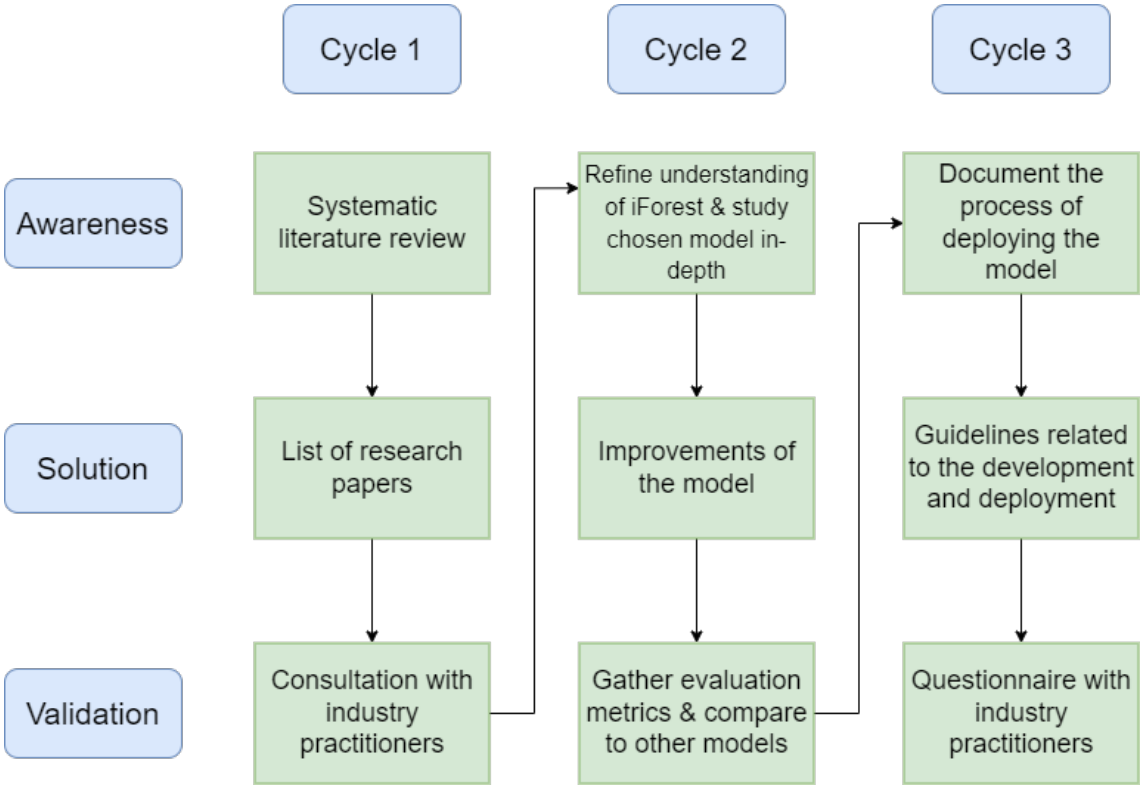

Figure 3.1: DSR feedback loop visualizing the workflow within each cycle.

Below is a description of each cycle and its expected outcome.

First Cycle (RQ1):

- *Awareness:* A lightweight literature was conducted, reviewing the existing literature on the iForest-based models adapted to handle data streaming scenarios. Following this, we determined a set of state-of-the-art models based

on iForest.
- *Solution:* The result of this cycle was a list of research papers on the iForest technique and its extensions.
- *Validation:* The first cycle was validated by consulting with our industry and academic supervisors to ensure the list of research papers provided a solid foundation for deciding a technique for our system.

Second Cycle (RQ2):
- *Awareness:* A refined understanding of the anomaly detection problem based on findings from the first cycle was acquired. The chosen model for anomaly detection was studied in depth.
- *Solution:* The anomaly detection model chosen from the first cycle was improved to suit the specific characteristics of the data we received from WirelessCar.
- *Validation:* A thorough evaluation of the enhanced model based on the AUC and F1-score evaluation metrics was conducted. These metrics have been used for model evaluation in several previous studies [8][23][24]. The scores were then compared to existing models. To be able to do the comparison we used real data from WirelessCar, which was manually labeled.

Third Cycle (RQ3):
- *Awareness:* The final model was deployed to AWS SageMaker, where it was continuously run and monitored. The process was documented to aid in formulating the guidelines.
- *Solution:* The result of this cycle was a set of guidelines on how anomaly detection can be integrated into the industry to aid software practitioners.
- *Validation:* The guidelines were validated through questionnaires at WirelessCar. These aimed to ensure the integration steps were aligned with what software engineers expect from such an artifact.

3.1 Cycle 1 - Systematic Review literature

In this section, the methodology used to answer RQ1 is presented. To gather a set of research papers related to iForest, a lightweight Systematic Review (SR) was conducted by following the guidelines detailed in [20]. The process involved three stages: search and selection, inclusion and exclusion criteria, and summarising the review's findings.

3.1.1 Search strategy

At the beginning of the search process, a search string related to anomaly detection using iForest was defined. The purpose of the search string was to find papers covering iForest applied to real-time data, as opposed to static datasets. To ensure all relevant papers were included, we included a few synonyms for "real-time data". The search string used in Google Scholar was **"anomaly OR outlier detection isolation forest" AND "real-time OR online OR streaming OR stream"** and the search string used in IEEE Xplore was **"All Metadata":anomaly detection) AND ("All Metadata":isolation forest) AND ("All Metadata":real-time**

OR "All Metadata":streaming data OR "All Metadata":data stream OR "All Metadata":online).

3.1.2 Selection strategy

To determine the relevance of the paper found by the search strings a brief inspection of the titles of the papers was done. This process involved comparing the title against the inclusion and exclusion criteria mentioned below. In cases where the paper's relevance could not be determined by the title, the abstract was glanced at to get a better understanding. In the next step, all abstracts were thoroughly read to filter out irrelevant papers further. If there was still uncertainty left after this step, the papers were partially read to understand their content's relevance better. Finally, all remaining papers were read through and summarized.

3.1.3 Inclusion and Exclusion Criteria

A key tool used in SR is inclusion and exclusion criteria, acting as a set of guidelines to help distinguish what papers are relevant to the purpose of the review. The criteria below were produced to isolate all papers that propose an extension to iForest used to detect anomalies in real time and meet additional criteria related to their credibility and relevance.
Each paper selected for the SR had to meet the following inclusion criteria:

- The study proposes an extension to Isolation Forest for real-time data.
- The study includes some of the most common evaluation metrics, such as AUC or F1-measure.

Conversely, the following exclusion criteria were used to determine if a study was irrelevant to our specific purpose.

- Exclude papers that are not published between 2019-2024. As anomaly detection is a rapidly evolving field, only articles published in the recent 5 years were considered relevant.
- Exclude papers that do not present a clear result in terms of evaluation metrics.

3.1.4 Validation for Cycle 1

After the list of relevant papers was compiled, the result was interpreted as a set of challenges with iForest highlighted in the papers. This was done to guide the development in cycle 2. To validate the list of research papers and the related challenges, an interview was done with a data engineer at WirelessCar who is experienced with ML in general. The questions were designed to validate that iForest is a good choice for the project, the papers chosen accurately reflect the body of research on iForest in streaming data, and the set of challenges identified align with what is seen in research. The questions are found in Appendix A.

3.2 Cycle 2 - Improved Isolation Forest Model

The aim of cycle 2 was to develop a model, with the knowledge gained in cycle 1, that effectively identifies anomalies in CloudWatch metrics data from a Kinesis stream. The research question addressed in this cycle was: RQ2:*Incrementing an existing artifact: How can the extended Isolation Forest algorithm be applied and tuned to perform well with metrics data from a cloud service?* To answer **RQ2**, the main activity of cycle 2 was a case study aimed at investigating the application of the Extended Isolation Forest algorithm for streaming data in cloud-based environments.

3.2.1 Case study design

For this thesis, we propose an ensemble ML model that applies the Extended Isolation Forest algorithm for anomaly detection in streaming data. The model was implemented and validated in a real-life scenario through the collaboration of the company WirelessCar. WirelessCar's infrastructure is predominantly deployed in the cloud, leveraging the services provided by AWS. Consequently, the data used in this case study was Amazon CloudWatch metrics from a Kinesis stream.

The study partly replicated previous studies when selecting model components but contained elements granting it some level of novelty. Similar approaches to anomaly detection have been taken, such as [19], where iForest, K-Means clustering, and Local Outlier Probability were combined for anomaly detection in streaming time-series data. One major difference is that it used the original iForest algorithm instead of the improved EIF algorithm. While using EIF supported by K-means clustering (KMeans) has been done in offline settings [12], we have not seen any applications in data streams, as research seems to lack any proposed models of EIF adapted to data streams. Additionally, to our knowledge, using power transform or other preprocessing algorithms to scale data before feeding it to iForest has not been done previously.

Figure 3.2 shows an overview of the case study context. The first relevant cloud service is a Kinesis stream, which is a vital component for the continuous process of ingesting data into WirelessCar's systems. Its main function is to collect and store data from customers. The operational health of a Kinesis stream can be monitored in CloudWatch, where several performance metrics are displayed. In the development stage, the data used was a dataset downloaded from CloudWatch consisting of historical data points dating a few months back represented by a set of metrics of the Kinesis stream. In the third service, SageMaker, the developed anomaly detection model was contained. After deploying it in SageMaker, the model continuously requested data from CloudWatch, which was processed in real time. The system predicted each data point, classifying it as normal or anomalous. Additionally, alarms were set up, notifying the company whenever an anomaly was predicted in the data stream.

The data was selected through consultation with software engineers at WirelessCar with domain-specific knowledge. Data quality assurance was mainly done by

inspecting the time-series graphs. The main criteria were to ensure there were no missing values in the data, enough historical data, and consistency in seasonal changes in the data. We manually labeled the static dataset of historical data by inspecting the individual plots of the metrics. We inspected 9 graphs using CloudWatch's built-in plotting tool during this process. Each plot consisted of 17200 data points, translating to 3 months of data with 5 minutes between each data point. We incorporated a few strategies to mitigate bias and ensure the labeling process was as accurate as possible. First, we labeled the dataset individually and then compared the results to ensure the same anomalies had been found. Next, we consulted with our supervisor, who has domain knowledge, to validate that the labeling was done correctly. To ensure the quality of the labeled dataset, this process was quite extensive. As a result, only one dataset was used to fit the time frame of the cycle.

The main validation of the model was done by evaluating it using different evaluation metrics. The metrics used were F1-score, AUC, and execution time. For the evaluation, the labeled dataset was used. To avoid overfitting, the dataset was split into training and testing data to ensure the model evaluation was done on data the model had not previously seen. The model was iteratively refined based on the performance observed during the evaluation. When the final model was implemented, these metrics were used to compare the model to other anomaly detectors. Due to time constraints, the comparison was conducted using publicly available models, as there was not enough time to implement any additional models from scratch.

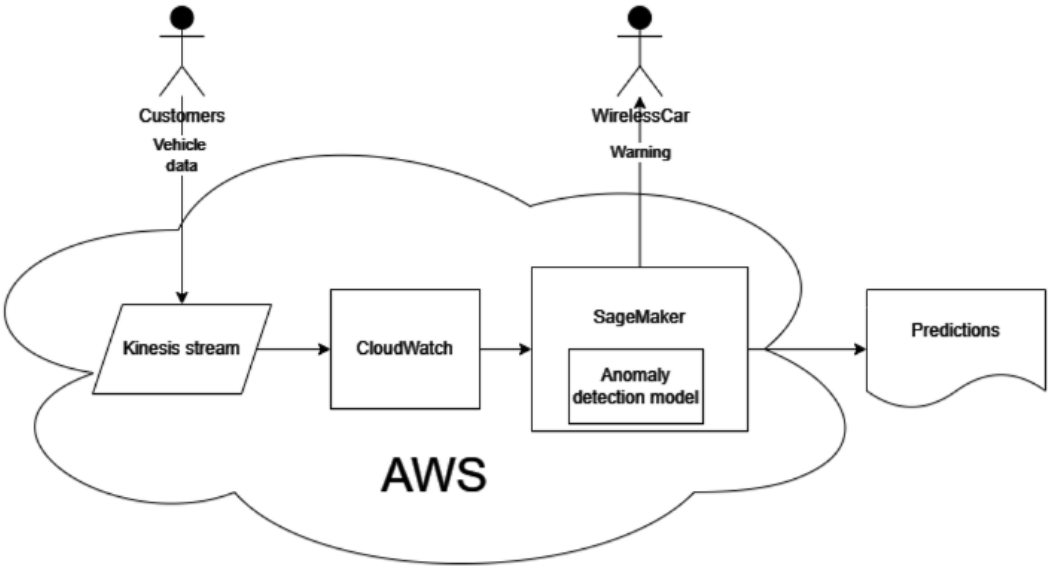

Figure 3.2: Overview of the case study context.

3.2.2 Data Analysis

The data used was CloudWatch metrics originating from a Kinesis stream. All of the features used came from the same Kinesis stream, which was selected after con-

sulting with our supervisor at WirelessCar. Some of the characteristics of the stream are that it has a high flow of data and has daily seasonal changes, which are the requirements we had for data quality assurance. Additionally, some features that were not particularly relevant to operational reliability were excluded. A total of 9 metrics were selected, summarized in Table 3.1.

Metric	Description
`IncomingRecords`	Number of records successfully put to the stream in the last period.
`IncomingBytes`	Number of bytes successfully put to the stream in the last period.
`GetRecords.Success`	Success rate of the `GetRecords` operation.
`GetRecords.Bytes`	Number of bytes retrieved with `GetRecords`.
`GetRecords.Records`	Number of records retrieved with `GetRecords`.
`GetRecords.Latency`	Time taken for each `GetRecords` operation.
`GetRecords.GetIterator-AgeMilliseconds`	Time elapsed since the last call of `GetRecords`.
`ReadProvisioned-ThroughputExceeded`	Measure that indicates that the amount of `GetRecords` calls exceeds the current throughput of cloud-based databases.
`WriteProvisioned-ThroughputExceeded`	Similar to `ReadProvisioned-ThroughputExceeded` but for the `PutRecords` operation.

Table 3.1: All CloudWatch metrics used as features in the static dataset.

A static dataset with the 9 selected features was downloaded from CloudWatch. The dataset included data points dating back 3 months, with the time interval between records being 5 minutes, resulting in a dataset with 17280 data points. Each feature was inspected separately to identify each metric's pattern. What was found was that the normal behavior of some features (`ReadProvisionedThroughputExceeded`, `WriteProvisionedThroughputExceeded`, `GetRecords.Success`, and `GetRecords.GetIteratorAgeMilliseconds`) was a constant value at 1 or 0. The other features followed a pattern with daily seasonal changes, where the values were higher during the day, especially before and after office hours. Additionally, a minor concept drift was noticed in the features with seasonal changes, where the amount of traffic in the Kinesis stream changed over time.

3.2.3 Proposed Model Overview

The proposed anomaly detection system is illustrated in Figure 3.3, where each box refers to a computational step that has been integrated into the system. The

boxes in the flowchart referenced as *Model Initialization* refer to the initial training of the power transform, EIF, and KMeans. The rest of the boxes in the flowchart refer to the real-time computational steps taken for anomaly detection. This can be summarized with the following steps:

1. A data point arrives in the model from the data stream. In the first step it is scaled and normalized by the power transform.
2. After undergoing the preprocessing, the data point is sent to the sliding window, where it is added, and the oldest one is discarded. It is also sent to EIF, where the anomaly score is calculated and sent to the KMeans layer and ADWIN.
3. ADWIN is updated and reports if a drift has been detected. If it has, the EIF is retrained with the data in the sliding window.
4. KMeans performs the final classification of the data point by assigning it to a cluster. The result is added to the output, consisting of a prediction list.

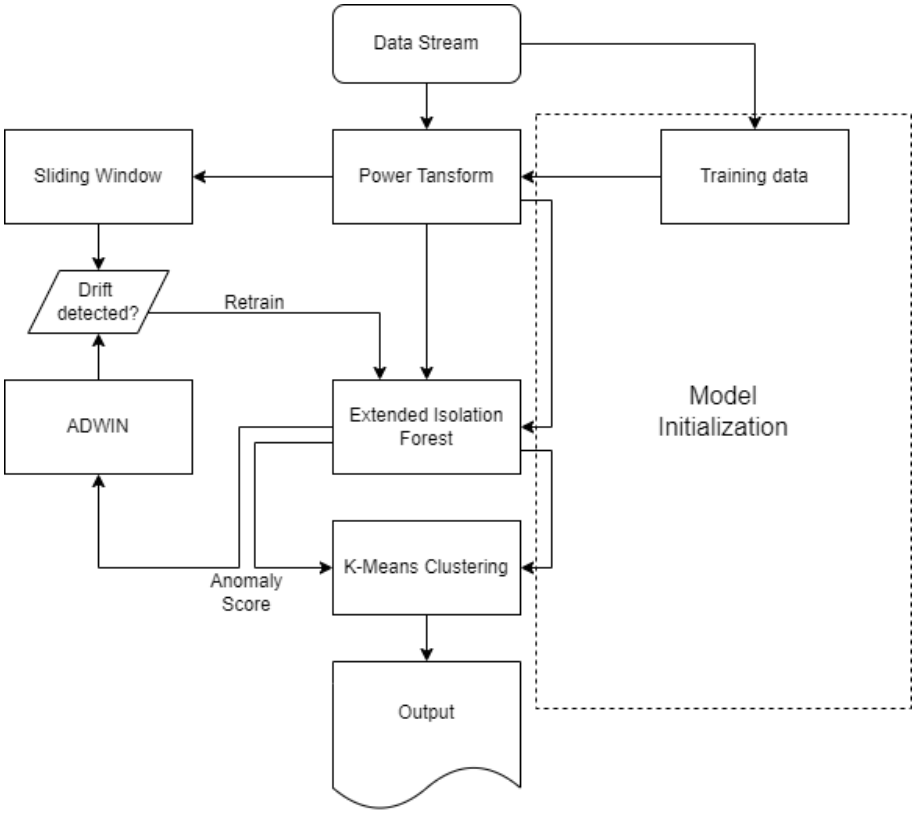

Figure 3.3: Flowchart illustrating the model

Power Transform

As seen in the 3.3, the first step in the model pipeline is to apply a power transform on the incoming data. A power transform is a tool used in data preprocessing to make the data distribution more Gaussian-like. This was necessary in our case as some anomalies in the same features significantly differed in size. This size difference resulted in the model's inability to identify smaller anomalies, so we introduced the

power transform.

The initial training phase of the power transform is essential as it needs to establish a baseline of how the incoming data is distributed for the transformations to be accurate. For simplicity, the number of data points used to fit the power transform was `window_size`, the same as for EIF and KMeans.

Once the training phase has been completed, the power transform is used to scale each new data point from the data stream, illustrated in the 3.3. In the same way as EIF, the Power Transformer undergoes retraining if the data exhibits concept drift.

Extended Isolation Forest

The Extended Isolation Forest, described in detail in Section 2.2.1, is the core of our proposed anomaly detection system. The EIF is responsible for producing the anomaly scores for each new data point in the system. However, an initial training phase is completed before the EIF model can begin processing data points from the data stream. This initial training phase involves building the model that produces the anomaly scores. As seen in the flowchart 3.3, the first part of the training phase is storing the first **n** data points that arrive in the system for training purposes. Once the desired amount of training data has been stored, which in this case equals four days' worth of data, the EIF algorithm selects a random subset of the training data, which is used to build the ensemble of iTrees.

Once the EIF has completed the initial training phase, it can process the incoming data in real-time. As illustrated in the 3.3, this phase begins after the data has been scaled by the Power transformer. During this phase, each new data point is traversed through all of the iTrees in the forest to compute its anomaly score. The anomaly score is derived by calculating the average path length from the iTrees root to the leaf node where the data point is isolated. The anomaly score is then passed to the KMeans algorithm to be classified.

K-Means Clustering

After the EIF algorithm has processed a data point and computed its anomaly score, it is used as input to the KMeans algorithm, and an anomaly label is determined. In anomaly detection, data points are differentiated by placing them in two categories, either normal or anomalous. Therefore, KMeans was initialized with two clusters, each corresponding to the normal and anomalous data. Using KMeans for translating anomaly scores into prediction labels has been done previously to address limitations of iForest [22] [19]. The limitation of iForest that this addresses is the need for an anomaly threshold to make predictions from anomaly scores. The original implementation of iForest provides a method for setting the threshold automatically. However, this introduces another issue: previous knowledge of the percentage of anomalies in the data is required. By instead passing the anomaly

scores to a clustering algorithm, the classification issue can instead be solved in an unsupervised manner with high accuracy.

The KMeans implementation used in the model is the one in scikit-learn [1]. The KMeans algorithm is used to divide a set of data points X into k clusters C with equal variance. In the first step of the algorithm, a data point is selected as the centroid μ for each cluster. The first centroid μ_1 is randomly selected, and $[\mu_2, \mu_k]$ is set to the data point furthest from the current clusters by calculating the maximum squared distance. When the clusters all have their initial centroid, each data point gets assigned to its closest cluster. After all points have been put in a cluster, μ is recalculated for each cluster. This process of dividing data points into clusters is repeated, starting at the assignment stage using the updated centroid. After each iteration, the change of the cluster center is measured, and the initialization stops when the centroids stabilize.

When the KMeans layer in the model is initialized, it first creates 2 clusters, one for normal data points and one for anomalies. The data that is used for initialization is the anomaly score output from EIF after scoring the training data. In the implementation by sklearn, there is a parameter `n_init` which determines how many times the algorithm should be run with different seeds, from which it chooses the run that minimizes the sum-of-squares within the clusters. Increasing this parameter can help identify reasonable cluster centers more consistently. As opposed to the EIF layer, the KMeans layer is only fitted during the initialization of the model and not during the retraining phase. Since KMeans is not directly modeled from the data, it is not affected by concept drift in the same way, and retraining only the EIF layer handles the impact of the drift for both layers.

Concept Drift Detection

Concept drift, as described in Section 4.1.2, is a common phenomenon that occurs in data streams. Concept drift refers to the change in the distribution of streaming data over time, which can significantly reduce the predictive capabilities of an ML model. In our case, we could observe that several of the data features exhibited concept drift; Figure 2.3 shows an example of this. We introduced a two-part solution to address concept drift's impact on the EIF effectively.

Firstly, to identify the drift, the anomaly detection system includes a drift detector, ADWIN, which is part of the river package available in Python. As illustrated by Figure 3.3, ADWIN uses the anomaly scores produced by the EIF as its input values. These scores are continually stored within a sliding window. This sliding window is then partitioned into two sub-windows, where the most recent anomaly scores are stored in one, and the older ones are stored in the other. ADWIN compares the statistical distributions of the two sub-windows and if there is a difference higher than a certain threshold, it reports that a drift has been identified.

Secondly, once drift in the anomaly scores has been detected by the ADWIN algorithm, which indicates that the statistical distributions of the incoming data have

changed, a retraining process of the model is initialized. This process is a critical part of the anomaly detection system as the model needs to be adapted to the new trends in the data to produce accurate anomaly scores. The retraining process first involves updating the current version of the power transform and then retraining the EIF model with the latest data stored in the sliding window.

3.2.4 Validation for Cycle 2

The validation process of Cycle 2 was done using an iterative approach. At first, an initial model based on EIF was created. Through multiple iterations it was then evolved, adding and evaluating new components over time to improve its performance. This was mainly done by gathering evaluation metrics, following the process described in section 3.2.4.

Evaluation Process

The evaluation process that was used throughout development will be described in detail in this section. It helped us identify flaws in the model and make critical design choices to improve model performance in the early stages of the cycle. In the later stages of development, it guided the fine-tuning of the parameters of different model components. For the evaluation, the main prerequisite was to have a labeled dataset. The dataset that was used was the one described in Section 3.2.2. As this was not yet labeled, we had to label it manually. This was done by inspecting each metric in the CloudWatch interface and recording the timestamp for each anomaly seen. A total of 58 anomalies were identified in this process.

The model was gradually incremented in the first iterations of the evaluation phase. That means that different components within the model pipeline were implemented and evaluated. Based on the results from Cycle 1, the main components investigated were dimensionality reduction, concept drift detection, and ensemble approaches. The dimensionality reduction component was excluded from the model, as it did not lead to any noticeable improvement in either execution time or accuracy. For concept drift detection, both ADWIN and KSWIN were considered. They were evaluated on the input data to EIF, the output data from EIF, and the final classifications. What was found was that the best setup was to use ADWIN on the output of EIF, i.e., the anomaly scores. The approach of using EIF combined with algorithms like KMeans and nearest-neighbor was tested. Both approaches were promising, but KMeans showed the best performance.

When evaluating the model in the later iterations, the focus was on fine-tuning parameters for different model components. The aim of this phase of the evaluation was to simulate how the model would work in a real-world scenario. To do this, the data was first split into training and testing data. For the training data, the first few days of the dataset were generally chosen to replicate the training phase after the model had been deployed. At the beginning of execution, the training data was used to fit the power transform layer, and the transformed training data was used to initialize the EIF. Finally, anomaly scores were calculated by EIF for each data

point and were used to fit the KMeans layer. In the next execution stage, each data point of the testing data was individually sent to be processed. For each data point, a prediction was made, classifying it as an inlier or outlier. To evaluate the performance, the metrics F1-score and AUC were used. They were chosen as they are commonly used for anomaly detection because they are unbiased for datasets with uneven distributions. Section 2.1.4 provides a detailed description of the evaluation metrics.

Different parameter values were evaluated during the phase where the model was fine-tuned. All other parameters were set to a default value, presented in Table 3.2. For the EIF layer, the parameters `ntrees`, `window_size`, and `sample_size` were evaluated. For the KMeans layer, the parameter `n_init` was evaluated. The model was run 5 times for each parameter. Due to time constraints, the parameters were only evaluated independently, while the others were static.

Parameter	Value
`ntrees`	50
`window_size`	1152 (4 days)
`n_init`	30
`sample_size`	288

Table 3.2: Default values that are used for the parameters when they are not being evaluated.

When the model was finished and evaluated, it was compared to other public models. The comparison was done by using the streamAD [25] repository, where several anomaly detectors for streaming data suited for both univariate and multivariate data are implemented. As the case investigated multivariate anomaly detection, the appropriate algorithms were used, namely: *Robust Random Cut Forest (RRCF)*, *Half-Space Trees (HST)*, *xStream*, *Randomized Subspace Hashing (RSHash)*, and *Lightweight on-line detector of anomalies (LODA)*. All of the models were tested with different input parameters, and the F1-score and AUC were calculated. In addition, the execution time for each model was recorded. Each model was initially trained using the first 4 days of data in the comparison experiment.

3.3 Cycle 3 - AWS deployment

The aim of cycle 3 was to evaluate the process of deploying an ML model for anomaly detection purposes in a cloud production environment. The research question addressed in this cycle was: **RQ3:** *How can the findings from the study provide guidelines that will aid software engineering practitioners in the process of deploying and integrating anomaly detection into a cloud-based workflow?* To answer **RQ3**, the model developed in cycle 2 was deployed in one of WirelessCar's AWS accounts using SageMaker. Section 3.3.1 describes the anomaly detection pipeline and its components, and Section 4.3.1 presents recommendations and guidelines aimed at

aiding software practitioners with implementing and deploying anomaly detection models in the cloud.

3.3.1 Anomaly Detection Pipeline

The process of building the anomaly detection pipeline for streaming data in AWS consisted of the components seen in Figure 3.4.

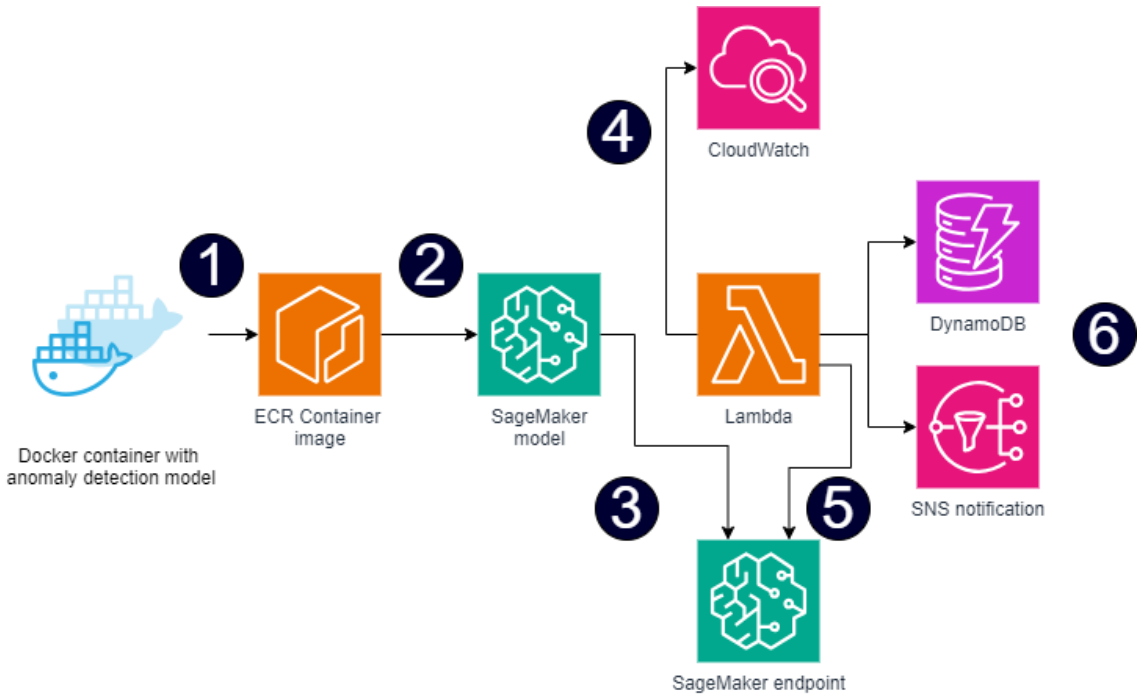

Figure 3.4: Overview of the steps taken to deploy the model in AWS and enable anomaly detection in data streams.

Each step in Figure 3.4 can be summarized as follows:

1. A Docker image containing the model is pushed to the Amazon Elastic Container Registry (ECR).
2. The model is created in SageMaker from the ECR image.
3. The model is deployed by creating a SageMaker endpoint.
4. A Lambda function is created and sends periodic calls to CloudWatch, requesting recent metric data.
5. After getting the latest data from CloudWatch, the Lambda function invokes the endpoint, sends the recent data, and gets a prediction returned.
6. If an anomaly has been predicted, a request is sent to Amazon Simple Notification Service (SNS), which sends an e-mail informing that an anomaly has occurred. The timestamp and the prediction are also stored in a DynamoDB table.

The following sections will provide a detailed description of the responsibility of each component.

Docker and ECR

Docker was a tool used to simplify the process of building, running, and deploying software. Docker does this by encapsulating all the necessary dependencies of the software in a so called container. By containerizing the software, developers can package the libraries, files, and other system tools needed to run the software in an isolated environment, essentially enabling the software to run on any host.

ECR is a fully managed container registry that can be used to deploy Docker images to AWS. It takes care of issues like storage, scaling, and management of containers. The service fits in well in the deployment pipeline, enabling integration between the Docker image and other AWS services.

As can be seen in Figure 3.4, the initial step in the anomaly detection pipeline was to create a docker image that contains the inference code of the model developed in cycle 2. This process included creating the files necessary for docker, such as the *Dockerfile*, *app.py* and *requirements.txt*. Once the docker image had been tested locally and was functioning as intended, it was pushed to Amazon ECR, allowing the image to be accessed by SageMaker.

SageMaker

SageMaker is a service used for building and deploying ML models in AWS. Section 2.3.2 provides a detailed explanation of the service. Our inference pipeline used it to create the model and the endpoint for the model. When creating the model, the docker image in ECR is referenced as the SageMaker access point. In the model creation, permissions for the model can be customized, setting what AWS resources it can use. After this, an endpoint can be created from the model. The creation of the endpoint begins with setting up its configurations. This step includes choosing an EC2 instance type. Instances vary in pricing, storage, and computational power. When this is done, the model is operational and can be accessed by invoking it.

Lambda

The AWS Lambda service runs a piece of code each time a predefined event is triggered. The trigger can be set up to respond to other services, like new data arriving in a data stream, or to call the Lambda function periodically. The output of the Lambda can be set to trigger new events, like calling another Lambda or sending a request to SNS to send a notification.

The Lambda function used in our pipeline is responsible for multiple actions. It has two different modes: model training and inference. The function is called every 5 minutes, as the anomaly detection model is initialized with that time interval. It first constructs a query and sends it to CloudWatch. This query is a request that contains all features that should be monitored and the name of the Kinesis stream to get the data. If the Lambda is in training mode, it requests 4 days of data. Otherwise, it requests the most recent data point to send to the model for inference.

When the response from CloudWatch is received, the Lambda extracts the data from the response. It then constructs a new query containing the data and a header describing if it contains training or inference data. This query is then sent to the SageMaker endpoint, and if the Lambda is in inference mode, the endpoint will return a prediction. If the prediction indicates an anomaly, the Lambda output will be forwarded to SNS to send an alarm.

SNS

The main function of SNS is to send notifications to connected users. These notifications can be sent in different formats, such as SMS, e-mail, calls to other services, or API calls. Events from other services are sent to SNS to initiate a notification, and in our case, this was done using Lambda. This final step of the deployment pipeline is important as it gives a way to handle unexpected events as they occur. Setting up SNS to notify a team working with the monitored data stream enables them to gain knowledge of potential issues faster and work on solving them. Currently, the SNS service is only set up to send an e-mail to a single instance for testing purposes, but this could be extended if the model is used in production.

DynamoDB

DynamoDB is a database solution offered as a service in AWS. DynamoDB is referred to as being "serverless", meaning that AWS fully manages its servers and infrastructure, allowing it to scale according to its demands automatically. DynamoDB is also a nonrelational database, which means it does not require SQL to manipulate the data. Instead, it uses a proprietary API to handle the interactions with the database.

In the current inference pipeline, DynamoDB stores the data points the model predicted as anomalies and their corresponding timestamps. By incorporating DynamoDB to store predicted anomalies and when they occurred, we facilitate a simple solution to analyze the anomaly detection model's predictive performance while providing a database of known anomalous data points. This can aid developers in their quality assurance and debugging activities.

3.3.2 Validation for Cycle 3

After finishing the model deployment in cycle 3, a set of guidelines was created. The guidelines were created to serve as general guidance for developing and deploying anomaly detection models in AWS. To validate them, two software engineers at WirelessCar were asked to answer a questionnaire. The questionnaire included general questions about the guidelines' correctness, comprehensibility, and usefulness. The questions are found in Appendix B

4
Results

This chapter reports and discusses the results of each cycle. It begins with the resulting list of papers on the state-of-the-art for iForest in data streams. Next, it demonstrates the evaluation of the model developed in the case study at WirelessCar. Finally, it lists all guidelines related to all cycles.

4.1 State-of-the-art of iForest in Data Streams

This section will begin by presenting the papers found in the search and selecting part of the SLR. The section then continues with the challenges of iForest found in these papers: Concept drift, dimensionality Reduction, and accuracy improvements.

4.1.1 Search and Selection Result

To identify all relevant research papers on iForest in streaming data, the methodology presented in Section 3.1 was used. Using the search strategy, Google Scholar and IEEE Xplore were searched. The search was done on 2024-01-29 and resulted in a total of 308 papers, 106 from Google Scholar and 202 from IEEE Xplore. After that, the selection strategy was followed. After applying the first selection step and reading the paper titles, 35 papers remained. Next, after reading each abstract, there were 15 papers left. Finally, after reading through each paper, 6 additional papers were excluded, resulting in a total of 8 papers in the final list. The result of the search and selection is presented in Table 4.1. From the models proposed in the papers, 3 have been made available in public GitHub repositories. However, due to errors in the codebase, the implementation proposed in *SP8* is not usable. Additionally, *SP1* is a continued work built on the model proposed in *SP2*, with the preceding model being included in the repository of *SP1*. However, the models in *SP1* (and *SP2*) are based on the library *Scikit-Multiflow*, which is deprecated. Consequently, there have been models found that are available for use.

4.1.2 Challenges of iForest

The systematic review resulted in 8 research papers being identified, each providing an extension to the original isolation forest algorithm to make it applicable to data streams. This section will highlight the main challenges with iForest seen in the papers and summarize how they were addressed.

Title	Reference	Year	Public Model
SP1: Anomalies Detection Using Isolation in Concept-Drifting Data Streams	[33]	2021	Yes*
SP2: Anomaly Detection for Data Streams Based on Isolation Forest Using Scikit-Multiflow	[32]	2020	Yes*
SP3: On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data	[17]	2021	No
SP4: Extending Isolation Forest for Anomaly Detection in Big Data via K-Means	[22]	2021	No
SP5: Real-Time Synchrophasor Data Anomaly Detection and Classification Using Isolation Forest, KMeans, and LoOP	[19]	2021	No
SP6: Fast Anomaly Detection in Multiple Multi-Dimensional Data Streams	[30]	2019	No
SP7: ASTREAM: Data-Stream-Driven Scalable Anomaly Detection With Accuracy Guarantee in IIoT Environment	[35]	2023	No
SP8: Anomaly Detection in Resource Constrained Environments With Streaming Data	[18]	2022	Yes**

*Built on the deprecated library Scikit-Multiflow.

**There is a public GitHub repository that cannot be used due to several code errors.

Table 4.1: List of scientific papers found in the SR, including title, citation label, year of publication, and accessibility of the proposed model.

Concept Drift

When applying iForest to streaming data, some challenges are highlighted in the literature. The first one is concept drift, which relates to changes in data distribution over time. To address the challenges of concept drift in continuously evolving data streams, a variety of methods have been proposed by researchers [17][30][32][33]. The most basic method to address these challenges is to use a fixed anomaly rate threshold, which updates the model if the pre-defined anomaly rate for a window of data points has not been reached[32][35]. The main limitation of using a static threshold is that tuning the parameter correctly can be challenging without prior knowledge regarding the distribution of anomalies. Another, more sophisticated approach to dealing with concept drift involves utilizing a drift detection algorithm. This has been done in the paper [33] where the authors compare the use of the well-known change detector algorithms ASWIN and KSWIN. ASWIN is an algorithm that modifies the size of the sliding window depending on whether concept drift has occurred or not. While the adaptive nature of ADWIN makes it a flexible choice for drift detection, it is ineffective for raw data [33]. KSWIN, on the other hand,

maintains a fixed sliding window and compares the distribution of the previous window to the current to detect concept drift. As it can be applied to input data directly, KSWIN is more sensitive to drifts than ADWIN. However, it is not as flexible due to the fixed window size. Page-Hinckley Test, which is very similar to KSWIN, has also been applied in [30] to mitigate the effects of concept drift.

Dimensionality Reduction

Another area of research related to iForest in streaming data is the concern for efficient models. The challenges of developing efficient models are mostly related to data exhibiting high dimensionality. With more dimensions in the data, finding meaningful patterns in the data becomes harder, and processing times increase. A common approach to solving this issue is to pre-process incoming data using principal component analysis (PCA)[30][18]. PCA is used to decrease the dimensionality of the incoming data, i.e., reduce the number of features that need to be processed. This is done by combining possibly correlated features into uncorrelated features called principal components. One limitation of PCA is that the principal components do not contain any meaningful information. A solution to this issue was proposed in the extension PCB-iForest$_{IBFS}$[17], where Isolation-Based Feature Scoring (IBFS) was used. IBFS is an unsupervised method that calculates feature scores for each feature during the iForest training phase, enabling dimensionality reduction while maintaining the interpretability of the features.

Accuracy Improvement

Other papers aimed to improve the anomaly detection accuracy of iForest. One way of achieving this was to use an ensemble approach, combining multiple ML techniques [19][22]. The main limitations of iForest are that it requires an accurate threshold for classifying anomalies based on their anomaly scores, and lack of information regarding the anomaly rate can reduce accuracy [22]. The method proposed in the ensemble models is to add a second ML layer using the K-Means algorithm, which is applied to the feature set produced by iForest, namely the anomaly scores. Doing this allows the data points to be more accurately partitioned into different feature sets. The extension proposed by [19] also includes a local outlier probability algorithm in the second layer of their model. By combining the outputs from the algorithms in the second layer, the model's accuracy was improved further compared to only using K-Means.

4.1.3 SLR Validation

The first cycle was validated through a questionnaire that we sent to a data scientist at WirelessCar. Using their knowledge and expertise, we got useful feedback and confirmation. From the questionnaire answers, the general feedback we received was that the findings were relevant, and that using iForest would be a suitable approach for the case study. After getting specific feedback, we included additional explanations about different drift detectors and their strengths and weaknesses.

4.2 Evaluation of the Improved iForest Model

This section begins by presenting the results of the evaluation of the model in a simulated streaming data setup. The evaluation metrics are then compared to a baseline represented by five state-of-the-art models for anomaly detection in streaming data.

4.2.1 Model Performance

The graphs presented here reflect the evaluation results where the parameter values are compared. Each table refers to a different parameter, and the included parameters are: `ntrees`, `window_size`, `n_init` and `sample_size`.

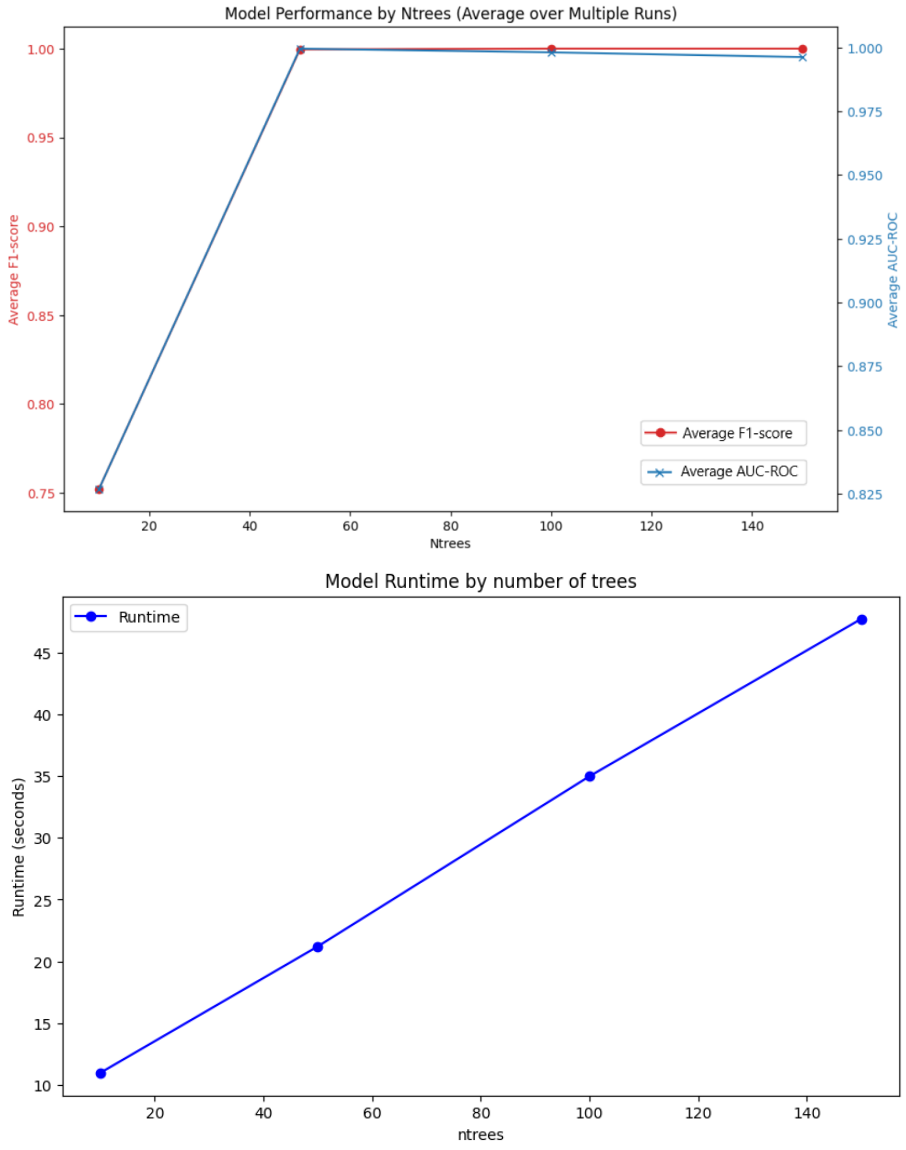

Figure 4.1: Performance metrics for `ntrees`. The values 10, 50, 100, and 150 were used for the comparison. The results indicate that the accuracy does not improve after increasing the number of trees above 50. The runtime of the model increases relative to the number of trees.

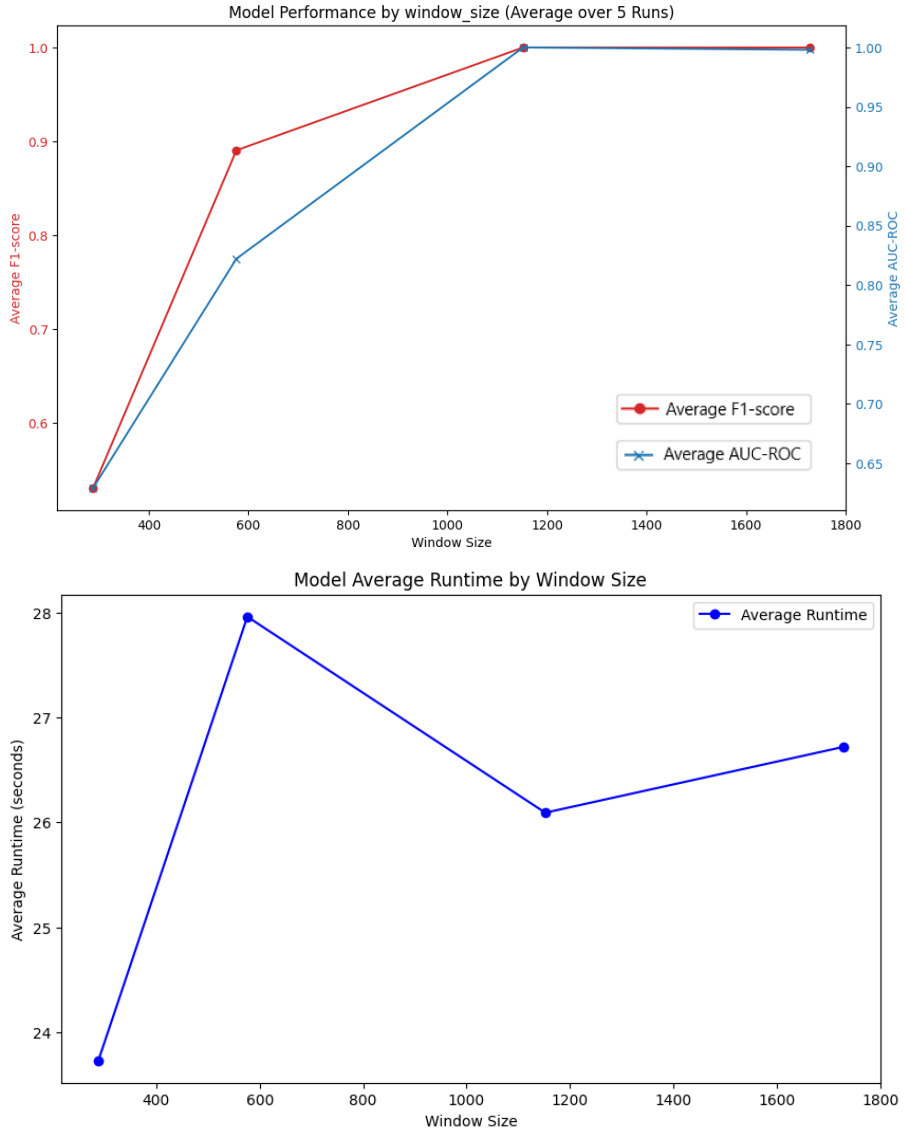

Figure 4.2: Performance metrics for `window_size`. Different multiples of 288, which is the number of data points in a day, were investigated. The values used were 1, 2, 4, and 6. The results suggest that there is an accuracy improvement until `window_size` is set to 1152, representing 4 days of data, at which point the performance stops increasing. The execution time is not affected significantly by the window size

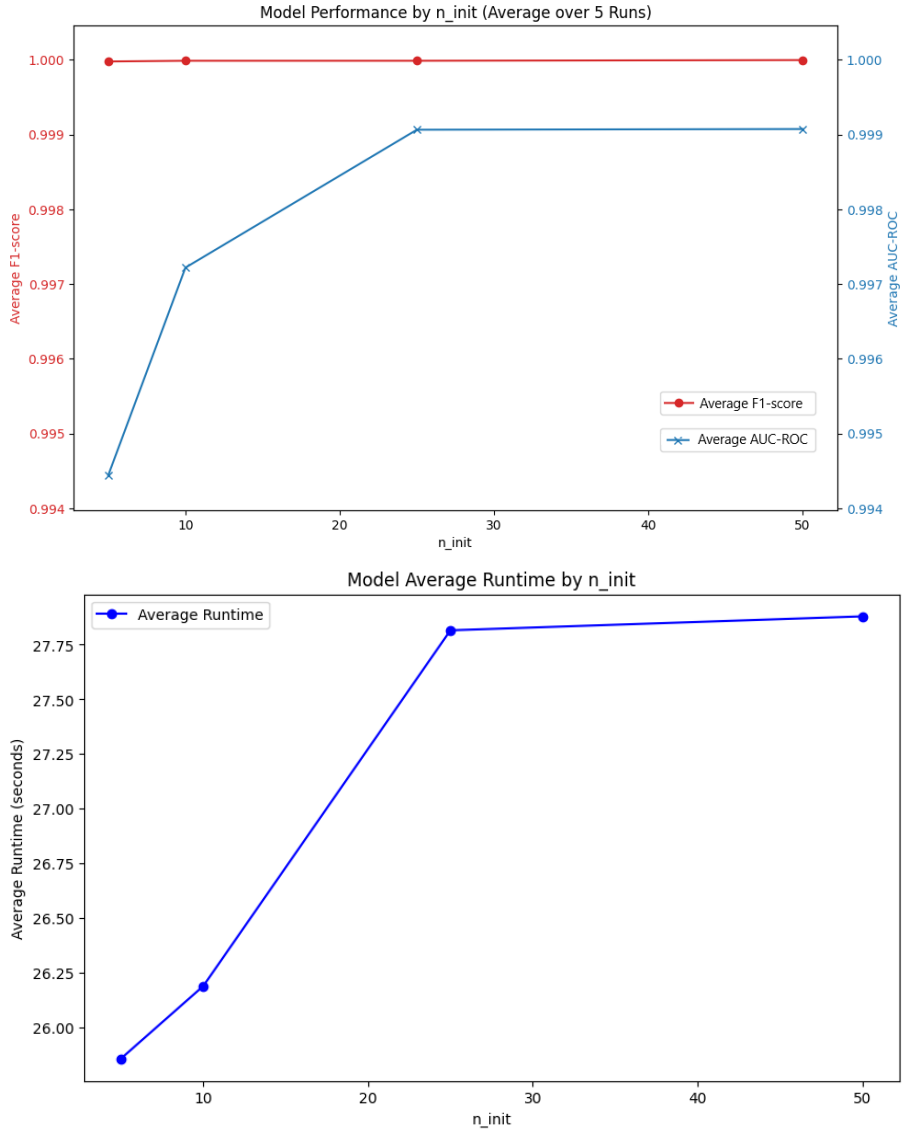

Figure 4.3: Performance metrics for `n_init`. The values 5, 10, 25, and 50 were used for the comparison. The results show the F1-score keeps a constant value for different metric values, while the AUC metric increases slightly with increased values. The runtime increases slightly with the increase of `n_init`.

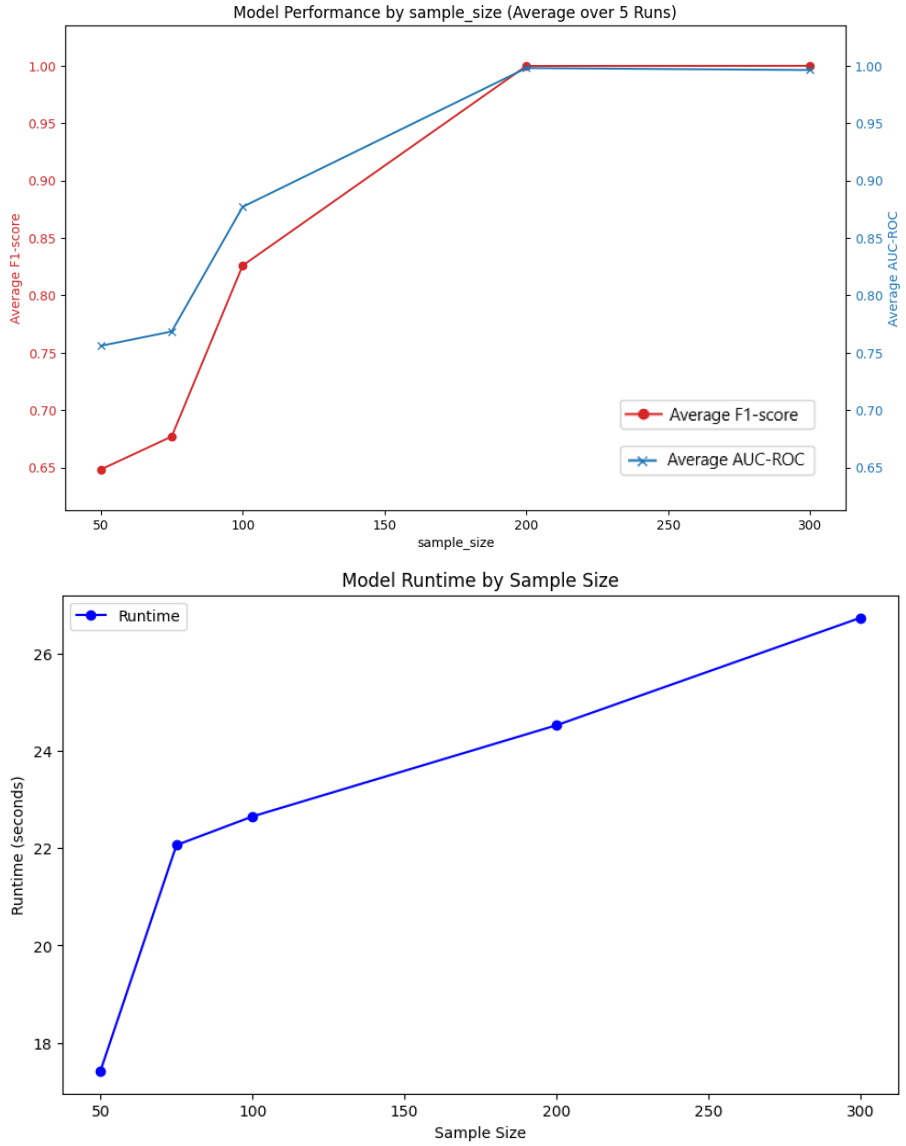

Figure 4.4: Performance metrics for `sample_size`. The values 50, 75, 100, 200, and 300 were used for the comparison. The accuracy increases with the number of samples until 200 samples are used. The execution time appears to be relative to the sample size.

4.2.2 Model Comparison

A part of the validation of the model was to compare the model to state-of-the-art anomaly detection algorithms. It was done to demonstrate the value of the developed model when applied to CloudWatch metrics data. For this comparison, the models in streamAD were used [25]. Each model was evaluated using a window size of 4 days, while other parameters, like the number of trees for tree-based algorithms, were adjusted between runs. Each model was run 5 times for each setting, and the average result is presented in Table 4.2. By doing this, each model was evaluated similarly to our proposed model to improve the fairness of the comparison. However, not all parameters were included, as it would require studying each algorithm deeper

and would extend the validation stage considerably. Therefore, the resulting metrics could potentially have been improved slightly.

Model	Settings	F1	AUC	Execution time (s)
RRCF	ntrees = 10	0.5	0.562	37.12
RRCF	ntrees = 25	0.5	0.517	92.52
RRCF	ntrees = 50	0.5	0.560	186.50
RRCF	ntrees = 100	0.5	0.572	374.78
HST	ntrees = 10	0.59	0.811	1.68
HST	ntrees = 25	0.57	0.828	2.74
HST	ntrees = 50	0.66	0.859	5.01
HST	ntrees = 100	0.68	0.883	9.29
xStream	nchains = 10	0.49	0.57	10.33
xStream	nchains = 25	0.49	0.655	20.08
xStream	nchains = 50	0.49	0.667	46.59
xStream	nchains = 100	0.5	0.665	94.82
RSHash	hash_num = 10	0.79	0.86	5.81
RSHash	hash_num = 25	0.76	0.904	10.74
RSHash	hash_num = 50	0.79	0.895	19.01
RSHash	hash_num = 100	0.79	0.889	34.97
LODA	random_cuts = 10	0.52	0.726	70.92
LODA	random_cuts = 25	0.5	0.782	171.69
LODA	random_cuts = 50	0.5	0.812	341.66
LODA	random_cuts = 100	0.5	0.809	686.37

Table 4.2: Evaluation metrics for running different anomaly detection models with different settings.

4.2.3 Discussion

This section discusses each parameter included in the evaluation and the value used for the final model. Finally, some conclusions will be drawn from the comparison with other models. Figures 4.1-4.4 show the performance metrics for different parameter values. The description of each graph and the choice of parameter values are listed below.

- **ntrees:** The `ntrees` parameter determines the number of trees, and increasing it can cause increases in accuracy, as seen in Figure 4.1. Since the anomaly score is given by the average path length of each tree, increasing this gives more consistency. This was seen in the evaluation, where there was a big increase from 25 to 50 trees. After that, no further improvements were seen as the F1 and AUC reached 1. Since the runtime increases linearly with the number of trees, it is reasonable to use the value that maximizes performance and minimizes runtime, which is 50.

- **window_size:** The `window_size` parameter determines the size of the sliding window, i.e., the number of recent data points stored and used for retraining of the model. The evaluation result for this parameter is displayed in Figure 4.2. Determining the optimal window size from the evaluation results is straightforward. The runtime appears to be unaffected by the size of the window, so choosing the value that maximizes accuracy is reasonable. The chosen window size is 1152, translating to 4 days of data. Even though a larger size could have been chosen using the presented reasoning, it is still better to use the smaller window size. In the case of a concept drift, using fewer data points for retraining ensures the model can adapt faster to the new data distribution, as a larger window will contain more outdated data points.
- **n_init:** The `n_init` parameter determines the number of times KMeans should be initialized with different seeds, increasing the likelihood of finding more optimized cluster centers. Using a smaller value can lead to the centroids being too similar, causing incorrect classifications. The results shown in Figure 4.3 suggest the parameter affects performance and that increasing it can improve accuracy. While the runtime increases slightly, it is only in the initialization step of the model, as the KMeans layer is not retrained when the model is online. As a result, a higher number of initialization runs for KMeans is preferred. The final value used for `n_init` is 50 to minimize the risk of poorly initialized centroids.
- **sample_size:** The `sample_size` parameter determines how many samples of the training dataset will be used to create each tree in the EIF. This value must be high enough to reflect the whole dataset's distribution accurately. However, as the graph of the runtime in Figure 4.4 suggests, a larger sample size leads to higher computational times. In the evaluation results, increasing the number of samples from 100 to 200 greatly improves performance. After that, higher values for the parameter do not show any improvements. As runtime increases with the number of samples, it is optimal to use 200 samples based on the results.

What can be concluded from the evaluation result is that for all parameters, the accuracy increases along with the parameter value until it has reached an F1-score and AUC score of 1. This means that all anomalies are found while no false predictions of anomalies are reported. However, it is essential to note that this result is based on the evaluation of a dataset that has been manually labeled, and the model is tailored specifically to the data. Manually labeling data may not be perfectly accurate as it is prone to human errors, and more complex anomalies could be missed. Additionally, it is not guaranteed that using a dataset with different characteristics would produce as good of a result.

Regarding the comparison with the other state-of-the-art models, the results are presented in Table 4.2. From looking at the AUC, it seems HST and RSHash perform best out of the five models. Looking at F1-score and execution time, this remains true as well. Out of the three metrics, the F1-score appears to be unreliable in the model comparison, as it usually is around 0.5, regardless of the AUC. This is because AUC is based on anomaly scores, while F1 is based on predictions. As the

predictions from anomaly scores are produced outside the models, the AUC metric is more representative of the accuracy of the models.

The results clearly show that the developed model outperforms all models within the comparison in terms of the performance metrics used. The HST and RSHash have relatively good metrics but low execution times. RSHash can score 0.904 for AUC in 10.74 seconds. HST scores above 0.8 in the same metric but is more efficient regarding execution time. These are acceptable scores, but when analyzing CloudWatch metrics data, the increase in efficiency is not required, as only one data point is handled every 5 minutes, and for that case, the increase is negligible. It is more important to correctly identify all anomalies, which is what our model manages to do in the test scenario.

4.3 Guidelines for Integrating Anomaly Detection in Cloud-Based Systems

This chapter presents the guidelines for integrating anomaly detection in streaming data using an extension of iForest. The process of synthesizing these guidelines involved careful consideration of all the cycles performed during the study. Firstly, the findings from the SLR helped us gain initial knowledge of the general requirements to consider when implementing iForest in a data streaming scenario. Secondly, the case study gave us a deeper understanding of how to construct an anomaly detection system based on the characteristics of the data used. Finally, the model was deployed in AWS to determine its effectiveness in a real-world setting, which resulted in insights into the AWS ecosystem and how different services can be connected. The guidelines aim to support software engineers with an overall knowledge of machine learning by providing recommendations regarding the activities that should be conducted during the development of an anomaly detection tool for streaming data.

4.3.1 Guidelines Overview

During the study, we identified three main stages in the development cycle of building an anomaly detection model for streaming data: *Data Analysis*, *Model Development* and *Cloud Deployment*, which are illustrated in Figure 4.5. For each stage, we have produced a set of guidelines covering the most important considerations to ensure a successful deployment. Additionally, some of the more general guidelines have a section presenting additional insights gained from the project.

Data Analysis: Here, we present guidelines that cover the data analysis activities that are performed to establish an understanding of the data and its patterns. The guidelines also include suggestions on how the data should be modified to be suitable for anomaly detection in streaming data using iForest.

Model development: These guidelines concern the important steps in the process

of transforming a model made for static datasets, in this case, iForest, into a model that is adapted to streaming data.

Deployment Stage: The final set of guidelines aims to guide developers in deploying a model in AWS. This includes giving an overview of what AWS services to use, explaining how to configure instances of services, and detailing how to prepare the model for deployment.

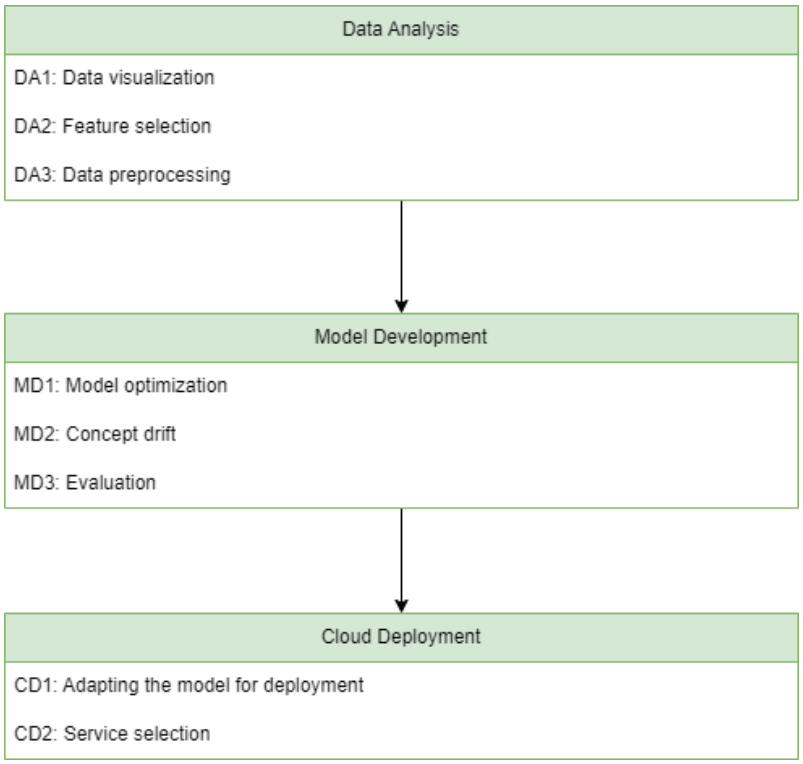

Figure 4.5: Overview of the proposed guidelines for different project stages.

4.3.2 Data Analysis Guidelines

This section will recommend activities to perform during the data analysis stage before starting model development. The recommendations are general and not specifically connected to the project's scope. However, some specific examples related to the strategies used are presented.

DA1: Data Visualization

From our experience, visualizing the data given to the anomaly detection system should be done as the first activity. Data can be visualized in multiple ways, but we found that plotting time-series graphs is the most informative. The data interpretation guides the whole model development and should be done early on in the process. Below, we list two main benefits of data visualization.

- *Identifying trends:* A benefit of data visualization is that it gives a visual interpretation of the data, which is necessary to understand the general patterns in

the data. Inspecting time-series graphs allows key characteristics like seasonal changes and concept drift to be understood. Seasonal changes can present themselves as the data distribution in some features changing depending on the time of day, and concept drift can be regarded as distributions changing over time. Understanding seasonal changes is necessary to ensure that the model does not deem expected changes in the data to be anomalous. Additionally, it is important to understand the unexpected changes that originate from concept drift and should be captured by the model to enable retraining.

- *Labeling datasets:* Apart from interpreting patterns, data visualization is necessary to make labeling datasets easier and more accurate. Anomalies are distinct in time-series plots as the graph usually follows a clear pattern, with anomalies appearing as distinct deviations from the trace.

Recommendations: The specific recommendation regarding data visualization is to do it as soon as possible. It gives a general overview of the data patterns, laying the foundation for design choices made throughout the development. Moreover, it enables labeling of the data, which is needed for iterative improvements and model validation. There are multiple ways to plot time-series data. Services like CloudWatch provide time-series plots within their console. Additionally, Python libraries such as Plotly Express can be used to create interactive graphs. For labeling time-series data, it is crucial first to study different types of anomalies, especially more complex ones like contextual and collective anomalies. For these anomaly types, it is recommended to consult with domain experts to ensure they are correctly captured.

Additional insights from the project: This stage of the project informed us that some features possessed a daily seasonality and that these features were prone to concept drift. This was important because the model needed to consider daily seasonal changes without interpreting them as concept drift. In the labeling phase, we also discovered that some anomalies were less obvious. In time-series data, contextual and collective anomalies are more complex than point anomalies. To properly label the datasets, we incorporated extensive consultation with a domain expert who could help us interpret the data and confirm the correctness of the labeling.

DA2: Feature Selection

Feature selection refers to the process of selecting a subset of the most valuable features from the original set of features. This process is considered highly important when developing an ML model. For anomaly detection in data streams, the feature selection activity is important for various reasons. In the setting of this thesis, we identified three main reasons why feature selection is important, highlighted in the bullet points below.

- *Reducing training time:* Most datasets include a large amount of data that can be considered redundant for the purpose of the model, and it does not provide any value in terms of predictive performance to the model. Given that time efficiency is of concern when developing anomaly detection for streaming data, any redundant data that may increase the processing time of a model should be removed.
- *Improving model performance:* Another reason for using feature selection is

that using a higher dimensionality dataset increases the risk of overfitting. This causes the model training to be too strict, leading to the model being unable to process new data appropriately. By excluding certain features with redundancy or lower importance, the model performance can be enhanced due to reduced overfitting.

- *Increasing interpretability:* Understanding the reason behind certain ML model decisions is important for two main reasons. Firstly, it can aid with debugging as it is important to understand how the model reacts to certain inputs to identify errors or biases. Secondly, it may increase confidence in the model by being able to explain and justify its decisions. We noticed that only having a smaller subset of features in our dataset made it clear to interpret and validate the model predictions.

Recommendations: During the development of the model, we could conclude that the most important suggestion for feature selection is to first consult with a developer with domain knowledge of the data. By utilizing their experience with the data, identifying redundant and irrelevant features can be done easier and more confidently. Also, chances are that their knowledge extends to insights about where different anomalies are generally present in the data and what features to include accordingly. Furthermore, various feature selection techniques used to find closely correlated features exist that can be used to reduce dimensionality. However, our suggestion is to be careful when doing that in anomaly detection. In the case where we used anomaly detection on CloudWatch metrics, we avoided doing this as several features were closely related but had unique anomalies seen in them.

Additional insight from the project: Due to the high complexity of the AWS infrastructure at companies like WirelessCar, AWS services can include hundreds of metrics in their CloudWatch monitoring, which was seen in the data stream used for our project. Thus, feature selection was not an easy task and had to be done by consulting with our supervisors, who had a deeper understanding of the AWS service and the metrics it produced. By incorporating someone with domain knowledge, we were able to determine the most relevant features much more quickly.

DA3: Data Preprocessing

The final guideline on data analysis is for data preprocessing. This relates to activities aimed at manipulating a dataset to make it more suitable for a model. It includes data cleaning, such as handling missing values and duplicates and transforming the data. We have identified two main recommendations for this guideline when performing preprocessing.

- *Data cleaning:* Data cleaning is necessary to ensure that the data processed by the model do not cause any errors. An important preprocessing step is to check that all data points have the same dimensions. This especially holds for a streaming data model, as errors caused by the input data can lead to the model crashing. Additionally, other issues like duplicated data points can introduce bias in the model.
- *Feature scaling:* Another preprocessing step to consider is feature scaling. Several feature scaling methods, such as normalization, standardization, and

transformation, can be used to map data points to new feature spaces. This step is not always necessary, but some ML models require some data normalization.

Recommendations: We recommend implementing functions that perform data cleaning on the incoming streaming data based on the model's specific requirements to ensure no errors occur during runtime. As we learned, handling missing values is crucial for model stability. Additionally, some data scaling techniques should be considered, especially if they are required by the chosen model. The inclusion of feature scaling highly depends on the ML model used and the characteristics of the data and should be implemented accordingly. While not common when using an iForest approach, we found that due to the characteristics of the data in CloudWatch, it can benefit from feature scaling.

Additional insight from the project: One crucial issue we noticed after the deployment of the model was the need for data cleaning. At the start of the deployment phase, we connected the model to a smaller data stream to test its functionality. We found that missing values in the incoming data stream could cause the model to crash. Therefore, we incorporated techniques to handle such unexpected behaviors to improve the stability of the model. Additionally, we identified the need for feature scaling. This is usually not required when using iForest. However, we found that some anomalies were harder to detect due to the characteristics of some features in CloudWatch. This was due to the training data including anomalies that deviated significantly more from the normal data than other anomalies. A power transform was included to scale all data points with a log-based transformation to address this. This caused anomalies to be more similar while maintaining separation from the normal data points.

4.3.3 Model Development Guidelines

The guidelines presented in this section will mainly address challenges that arise during the development phase of an anomaly detection model for streaming data. As a general recommendation for the model development, we suggest using Python, as it supports all libraries used for development. These guidelines are closely related to the case study and the lessons learned. However, some insights are more general and related to characteristics of time-series data and model evaluation rather than the model developed.

MD1: Model Optimization

There are multiple ways to optimize an anomaly detection model to improve its prediction accuracy and performance. Some of the most impactful variables include parameter tuning and the inclusion of additional algorithms to improve the model. Here, we present some of the important things to consider during development to optimize the model.

- *Component selection:* The first step in constructing an anomaly detection model is to choose an appropriate algorithm to base the model on. An ap-

proach that combines multiple algorithms can also be used to address limitations within the chosen model.
- *Parameter tuning:* Model tuning has a big impact when it comes to optimizing the model. It involves changing the model's parameters to find the best trade-off between accuracy and efficiency.

Recommendations: After developing an iForest-based model, we have identified some general recommendations for future practitioners interested in developing a similar model. Firstly, choosing EIF over iForest is recommended as it potentially improves accuracy without introducing higher computational requirements, as explained in Section 2.2.1. Secondly, combining EIF with KMeans enables the model to make accurate classifications instead of outputting anomaly scores. Thirdly, analyzing the data can be helpful when choosing parameter values. For this thesis, for instance, the data included daily seasonal patterns. To ensure the model was trained on enough data to capture this behavior, the sliding window size was set to 4 days of data.

MD2: Concept Drift

In the SLR conducted during cycle 1 of the project, concept drift was identified as one of the main challenges when developing a model for anomaly detection in streaming data. Concept drift refers to the change in the data distribution over time. During the project, we identified two main challenges when addressing concept drift in the data, and they can be expressed in the following way:

- *Identifying drift:* The first step in mitigating the effects of concept drift is to identify when it has occurred accurately. This is not a trivial task, as considerations regarding which data needs to be analyzed for drift are difficult. There are mainly two models used for drift detection: ADWIN and KSWIN. ADWIN uses an adaptive window that changes based on the behavior of the data and calculates its variance compared to historical data. KSWIN, on the other hand, has a fixed window size and calculates drifts using a statistical test. As a result, KSWIN has the potential to detect drifts more precisely, but due to its fixed window size, it is more suitable for batch anomaly detection. ADWIN can be more useful in streaming data due to its adaptive window. Additionally, apart from being applied directly to the raw feature values, ADWIN can be used by either the model predictions or anomaly scores produced. This way, the model can potentially react to more subtle changes in the data.
- *Reacting to drift:* A concept drift can make the model outdated as it has not been updated to accommodate the new data patterns. Therefore, managing the effects of concept drift is an important activity for any anomaly detection model for streaming data, as this will help the model maintain its predictive performance. This is generally done by storing a list of the most recent data points and using them to retrain the model if a drift is detected.

Recommendations: During the project, both of the drift detection algorithms mentioned in the literature, KSWIN, and ADWIN, were tested, and it was concluded that ADWIN performed better. This is likely due to the adaptive windowing in ADWIN, which is better suited for processing one data point at a time. Therefore, our recommendation to the developer is to use ADWIN for drift detection in

streaming data. Additionally, we evaluated when the ADWIN algorithm performed the best and could conclude that the recommended data to use is the anomaly scores produced by the EIF when there is seasonality in the data. This is because the anomaly scores do not change depending on the time of the day. Thus, it avoids being affected by daily seasonal changes.

MD3: Evaluation

A core part of the development cycle of a project is evaluation, as it enables iterative improvements and validation of the model. The model is evaluated by measuring different evaluation metrics such as F1-score and AUC-ROC. The following should be considered when evaluating a model for anomaly detection in data streams:

- *Measure performance:* To evaluate any ML model, evaluation metrics need to be used. For anomaly detection, the most common metrics are F1-score and AUC-ROC. They are well suited for data with class imbalance, such as in the case of anomaly detection. These metrics are used to measure how well the model correctly identifies anomalies.
- *Check adaptability:*When applying anomaly detection on data streams, there are some additional challenges to consider during the evaluation process. The main challenge is evaluating the model's adaptability to concept drift, which essentially refers to the model's ability to respond to distribution changes in the data over time. It is highly important to ensure the model can handle concept drift; otherwise, it will gradually lose its predictive performance over time.
- *Identify different anomaly types:* During the evaluation, it is important to ensure the model can detect various types of anomalies. While the most common type is the point anomaly, which should be present in any dataset containing anomalies, it is also valuable to verify that the model can detect contextual anomalies.

Recommendations: Evaluating an anomaly detection model for streaming data requires careful consideration, and some concrete suggestions will be presented here. To begin with, the model must be created to handle one data point at a time. Even though a static dataset is used for evaluation, it should be looped through to reflect the behavior of streaming data. Moreover, the dataset should be plotted and inspected to localize where concept drifts are seen in the data. As the model runs, it should log every time a drift is detected, and the model should be retrained to ensure it adapts to distribution changes in the data properly. A way to determine if the model can detect drifts if no drifts are present in the dataset is to manually inject drifts by scaling values in different places in the dataset. Lastly, it is recommended to check that the model can find anomalies of different types. Since some anomaly types are less usual, injecting anomalies in the dataset can be a good way of checking how well the model handles those.

Additional insights from the project: When evaluating the model, we manipulated the dataset to test the model against certain conditions. Firstly, we introduced an abrupt concept drift and validated that it was found. As the dataset only included drifts occurring over a longer time period, this was done to ensure it could

find other types of drifts. Secondly, we introduced some anomalies, like contextual ones, to validate that the model could also identify them.

4.3.4 Cloud Deployment Guidelines

The guidelines presented in this section will give recommendations when deploying a model in AWS. The guidelines related to cloud deployment are more case-specific than previous recommendations. This is because they are specifically connected to deployment in AWS. However, some guidelines, like those related to the containerization of the model, are not specific to the case of AWS.

CD1: Adapting the Model for Deployment

Before deploying the model, it must be adapted to work correctly within the cloud workflow. The deployment process takes a while to complete, and once the model is deployed, it cannot be changed. Some general points to consider before deployment are listed below.

- *Docker:* To be able to deploy a custom ML model in the cloud, in this case, using AWS SageMaker, it is generally required to containerize the software. SageMaker provides a variety of pre-built containers for typical ML tasks, but if the inference code depends on any custom liberties or frameworks, a custom container needs to be built.
- *Web application:* When deploying a model as a SageMaker endpoint using a custom Docker image, it is required to include a web application. The purpose of the web application is to handle communication with the model. The web applications enable the SageMaker endpoint to process the incoming data and return the predictions.
- *Logging:* Logging is valuable for anomaly detection models as it allows for monitoring of predictions and other metrics. It is essential to carefully consider where to use logging before deploying a model in SageMaker, as it cannot be reconfigured later.

Recommendations: Before deploying a custom anomaly detection model, developers should consider the following recommendations: Firstly, if the inference code contains custom libraries or frameworks, as was the case for our model, the model needs to be containerized with all of its dependencies. The container image should be carefully tested locally before being pushed to ECR, as it can not be modified once stored. If the testing has been done thoroughly, it will significantly speed up the rest of the deployment process. Secondly, the container requires a web application that handles the client and model communications. The web application should include at least two functions: one for initializing the model and one for making predictions. Similarly, the model should have methods implemented that correctly handle this. Finally, there should be enough logging statements to ensure proper model monitoring. We recommend at least using logging when a drift is detected and when an anomaly is predicted.

Additional insight from the project: The proposed recommendations are directly derived from lessons learned during the model deployment. We had to deploy

it a few times before it worked correctly, and the main difficulties we faced are reflected in the recommendations. A major issue was that the model lacked sufficient logging, causing debugging to be more difficult and the re-deployment to take multiple iterations, which is why logging is so important.

CD2: Service Selection

Deploying anomaly detection models in AWS can be done in multiple ways, and generally, all approaches involve using SageMaker. We suggest using the approach detailed in Section 3.3.1 for deploying custom models. Here, we give further recommendations related to different phases of the proposed pipeline.

Recommendations: The suggestions for service selection are related to the anomaly detection pipeline used in this thesis. Below is a list of recommendations for designing and configuring a pipeline.

- *Retrieving the data and making prediction:* We recommend using Lambda to handle communication between the model and the data stream. It can be set up to react to certain events. In the case of fetching data from CloudWatch, the straightforward approach is to call the function after a predefined time interval. In our model, the metrics fetched are for the latest 5 minutes, so the Lambda function is called every 5 minutes. The function should then include a call to the SageMaker endpoint. The data point is processed in the endpoint, and a prediction is returned to the Lambda function. Lambda facilitates the use of several programming languages. In our Lambda function, Python was used, but the selection is up to personal preference.
- *Reporting and storing predictions:* When deploying a model that makes real-time predictions, it is also important to have a way of reporting anomalous predictions. AWS supports different ways of doing this, but we recommend utilizing SNS, which can be quickly set up to send an alert, such as an email, once an anomaly is predicted. Additionally, for debugging purposes, it can be valuable for the developers to set up a database where each predicted anomaly is stored. AWS offers several options for this, but the recommended service is DynamoDB. Due to its pay-per-use pricing model, it works well for small storage requirements with infrequent reads and writes, such as in this case.
- *Monitoring cost and performance:* When deploying a model as an endpoint on AWS, developers should be mindful of the cost associated with hosting such an endpoint. It is important to consider the requirements of the model to properly determine the instance type needed to support the model's operations. Instance types come in varying computing power and memory, and the cost can differ considerably between them. Our suggestion is to choose carefully to ensure the cost is minimized while the instance meets the requirements of the model. To validate the choice of instance type and monitor the model's general performance, AWS CloudWatch can be configured to keep track of the endpoint's CPU utilization, latency, and memory usage. By inspecting the graphs provided by CloudWatch, developers can conclude if the model is working as it should.

4.3.5 Guidelines Validation

Validation of the guidelines was done through questionnaires, where participants were asked to read the guidelines and answer related questions. The participants were two WirelessCar employees with different experiences working in different company areas. The general impression participants reported was that the guidelines are structured and easy to follow, giving a clear overview of the deployment process in AWS. They mainly touch on the surface of the process, and to actually develop and deploy a model, more research would be required by individuals following the guidelines. However, participants consider this to be an appropriate level of detail for the purpose of the thesis as it gives enough guidance and direction.

Some specific feedback points and actions taken to address them are presented below:

1. **Points or terms that were unclear:**
   Feedback: There was confusion regarding the statement: "Firstly, choosing EIF over iForest is recommended as it potentially improves accuracy without introducing higher computational requirements." The respondent noted a lack of explanation on how or why EIF can improve accuracy.
   Solution: While the explanation can be found in earlier chapters, it is unclear to someone reading only the guidelines. A reference pointing to the EIF description was added to improve clarity in this statement.
2. **Missing information:**
   Feedback: The guidelines do not mention the programming language or libraries used, which would be beneficial, especially for the ML model and Lambda functions.
   Solution: To improve clarity regarding this, recommendations for programming language selection were added.

5
Discussion

This chapter discusses the results presented in Chapter 4 by answering the research question posed in the introduction. Then, we discuss the internal, external, and construct threads to the validity of our study and the measures we took to mitigate. Finally, the chapter is finished by discussing the future work that can be done for the study.

5.1 Discussion of Research Questions

In this section, the research questions will be discussed. This will be done by presenting a summary and an interpretation of the outcomes of each cycle.

5.1.1 Answering RQ1

RQ1 - Current state of anomaly detection in software testing using iForest: *What are the main challenges with iForest in streaming data, and what state-of-the-art models are currently available?*

This research question essentially has two parts: identifying challenges with iForest and finding models that are ready for use. Both of them were addressed through an SLR. The SLR mainly aimed to outline the state of research on iForest in streaming data. It resulted in a list of 8 highly relevant papers that present novel anomaly detection approaches were identified. To answer the first part of **RQ1**, we interpreted the findings to produce a list of three main focus areas: *concept drift*, *dimensionality reduction*, and *accuracy improvement*. In Section 4.1.2, these challenges are described in depth, and the proposed solutions are summarized.

To answer the second part of *RQ1*, each paper in the research list was searched for related repositories. This revealed a lack of publicly available repositories related to the papers. Only 3 papers had a corresponding repository, but none was ready for off-the-shelf use. The repository presented in *SP8* had several errors in the code, making it too difficult to use. The one related to *SP2* was outdated as it preceded the repository in *SP1*, which included *SP2's* model. Finally, the repository in *SP1* was based on the deprecated library *Scikit-Multiflow*. In conclusion, no state-of-the-art models that were presented in the research were ready for use. However, since the code is public for some models, they could potentially be modified to be usable. The research papers without public code bases also presented pseudo-

code and detailed model overviews. Therefore, it is possible to replicate most of the studies identified in the review. Due to the time-constraint of this project, it was not done.

In summary, we believe our first research question has been answered. We have highlighted the challenges of continuously evolving data by analyzing the currently available research on applying iForest to data streams. Such challenges include concept drift and efficiency demands and how iForest can be extended to overcome these. Additionally, we have searched for state-of-the-art models presented in the research. We have found that for all papers, either there was no code available or the code was outdated and cannot be used.

The SR's findings strongly indicate that iForest has the potential to efficiently identify anomalies in a streaming setting. We have validated the findings with an industry professional who agreed that our proposed approach was reasonable. Consequently, we confidently chose the iForest algorithm to implement the anomaly detection system.

5.1.2 Answering RQ2

RQ2 - Optimizing iForest for CloudWatch metrics: *How can an extension of iForest be refined and tuned to perform well with metrics data from a cloud service?*

This research question explored how an extension of iForest, in this case, EIF, can be tuned and applied to data in a streaming context. To address this, we conducted a case study, which entailed the development of an anomaly detection model using the knowledge gained during the SLR. The goal of the research question was not necessarily to craft a new algorithm but rather to adapt and combine existing technologies for streaming data. The research question essentially consists of two sub-questions. Firstly, what must be done for EIF to be applied to streaming data? Secondly, how can EIF be optimized and adapted to the specific case of finding anomalies in CloudWatch metrics data?

The first part of the research question relates to combining additional components with EIF to make the algorithm applicable to streaming data. This included adding a sliding window for storing recent data points. Furthermore, the concept drift detector ADWIN was introduced to monitor pattern changes in the data. These two components are required for anomaly detection for data streams as they ensure the model can adapt to shifting distributions in the incoming data.

The other part of the research question, the optimization, addresses the limitations of EIF, both general limitations and those specific to the case study. One of the general challenges of EIF is the classification of anomalies, as the model only predicts an anomaly score. For the model to interpret the score, the rate of anomalies in the data is required. This was solved by applying the clustering algorithm KMeans on the scores to separate anomalies. A limitation related to the case study

context was the model's inability to identify certain anomalies. Due to the characteristics of the data, a power transform was used to preprocess incoming data to improve the model's detection accuracy.

The evaluation process has shown that applying the model to CloudWatch data from a data stream outperforms the baseline in different performance metrics. Using a dataset from a data stream in production, the model could correctly identify all anomalies without any false predictions. This implies that the model is a powerful tool for monitoring anomalies in CloudWatch metrics data. However, this result may not extend to other data with other characteristics as the model was only evaluated on one dataset. Further validation of the approach would require placing the model in a different context.

Throughout cycle 2 and by conducting a case study at WirelessCar, we argue that RQ2 has been thoroughly answered. The case study included exploring different ways to adapt models to the requirements of anomaly detection in data streams. Moreover, we have shown how EIF can be tailored according to the data seen in CloudWatch to improve its performance. By evaluating the model thoroughly, we have fine-tuned its parameters to ensure it is fully optimized to deploy for anomaly detection on data streams in the cloud. While we cannot be certain whether the promising evaluation result would extend to other domains, it must be left as the topic for future work. After all, the goal of the case study was to use the model in a specific context, and we believe that has successfully been done.

5.1.3 Answering RQ3

RQ3 - Guidelines for integrating anomaly detection in the industry: *How can the findings from the study provide guidelines that will aid software engineering practitioners with deploying and integrating anomaly detection into a cloud-based workflow?*

In cycle 3, we focused on formulating a set of guidelines based on the insight gained from previous cycles to address RQ3. Additionally, cycle 3 involved deploying the anomaly detection model in a cloud-based infrastructure and documenting key challenges with the process. The knowledge and experience accumulated from each cycle have been summarized in recommendations of three main categories: *data analysis*, *model development*, and *cloud deployment*. Each of these categories mainly corresponds to the main findings of each cycle.

The SLR and case study findings were used to synthesize data analysis and model development guidelines. By grounding the development phase in the theory found in the research, we explored different techniques to approach anomaly detection in data streams and ways to overcome the challenges of it. Recommendations related to data analysis and model development have been written from this. Furthermore, by integrating the model into a cloud-based infrastructure, we have acquired knowledge that has been leveraged to produce the final set of guidelines. These are aimed

at helping practitioners adapt models for cloud deployment and make the process more seamless.

Our recommendations have been validated through industry feedback, ensuring their relevance, comprehensibility, and applicability. We intend for these guidelines to serve as a framework for practitioners and researchers interested in developing and deploying anomaly detection models. We believe the guidelines capture the general challenges of this process, and thus, we claim that the final research question has been answered.

5.2 Threats to validity

This section will present the identified threats to validity in terms of external, internal, and construct validity. External validity refers to the level of generalizability that one can claim from the results or conclusion of the study. Internal validity is related to the threats within the study context that can discredit the cause-and-effect claims. Construct validity refers to the degree to which the methods used for evaluation are created to measure what they are intended to measure correctly. Additionally, the strategies incorporated to mitigate the effect of the threats are discussed.

5.2.1 External Validity

The main threat to the external validity of this study is that the findings are related to the specific context of the case study. This is due to the model being evaluated on only one dataset and that the guidelines is a result from the case study. Therefore, there is a risk that the guidelines are too narrow and focused on AWS deployment and iForest development specifically. Due to this, some of the guidelines contain specific information related to the case. With this in mind, we have tried to construct some of the recommendations to make them applicable in a broader context.

5.2.2 Internal Validity

To ensure different biases are avoided when validating different findings, we analyzed potential threats to internal validity and made strategies to mitigate them. One identified threat relates to the final guidelines' evaluation process. It originates from the bias that could be introduced when validating the deployment process with software engineers who participated in it. To mitigate the bias, we ensured the deployment process was done independently, without external help or guidance from our supervisors at WirelessCar. Additionally, to avoid bias further, we validated the guidelines with different people at the company.

5.2.3 Construct Validity

A labeled dataset was needed to evaluate the anomaly detection model. As no such datasets were available, a dataset needed to be labeled manually by inspecting

the data for each feature and noting any visual outliers. By doing this, we captured all point anomalies in the dataset, but more complicated ones, like contextual anomalies, might have been missed. We identified this potential bias as a threat to construct validity. As we were unfamiliar with the data, we consulted our industry supervisors to reduce the impact of this threat. To their knowledge, no particular complex anomalies had previously been seen in the data.

Additionally, the short time period of the deployment cycle makes it hard to validate how well the model performs over a longer time-period. As a result, measuring how well the model adapts to concept drift becomes impossible. To validate the model's adaptability, the static dataset was used. We could see that the model correctly reacted and retrained when the data changed. However, a longer monitoring window would be required to validate it in the deployment context.

5.3 Future Work

The research conducted in this thesis aims to lay a foundation for integrating anomaly detection systems in cloud-based environments. However, several areas can benefit from further exploration to enhance the effectiveness and applicability of the proposed anomaly detection methods. The following points outline potential directions for future research:

Deployment in diverse cloud environments: The guidelines relating to cloud deployment are based on the experience of deploying a model in AWS using monitoring data from CloudWatch. Future research could diversify and generalize the guidelines by exploring deployment in other cloud environments.

Longitudinal monitoring of the system: One of the threats to validity that were identified in the study was the difficulty of validating the deployed model's ability to react to concept drift. While this was analyzed on a static dataset, it could not be done for the deployed model as it would require the model to run for an extended period, which was not feasible in the project's scope. Consequently, a topic for future research would be to monitor the model over a longer time to measure how well it adapts to shifts in data distribution.

Using the model with diverse data: During the case study and deployment process, only one dataset and set of metrics were used. The model performed very well for the given data, but the result cannot be generalized without further evaluation using data with differing characteristics. A potential task for future work is evaluating the model with other datasets and cloud services.

Comparing the model with other iForest extensions: In the model evaluation, our proposed system was compared to other anomaly detectors by calculating different performance metrics. While the result was promising, none of the evaluated models were based on iForest. This was because none of the identified state-of-the-art iForest extensions for streaming data were publicly available. However, most

papers presented comprehensive descriptions and pseudo-codes for the algorithms, making the implementation reproducible. As this was impossible in this project, we leave that as a suggestion for future work.

6
Conclusion

This thesis explored the possibility of applying the algorithm Extended Isolation Forest for anomaly detection in data streams. The proposed model was developed and evaluated using CloudWatch metrics data from a Kinesis stream. This resulted in a model that could detect all visual anomalies found in the acquired dataset. Additionally, the final model was deployed in one of WirelessCar's AWS accounts, and the entire process covering data analysis, development, and deployment was compromised as a set of guidelines. Given the promising results of the model and guidelines, the study still faces some limitations regarding its generalizability as it was conducted in a specific setting with a specific purpose, and these limitations have been summarized and suggested for future studies. Regardless, we have successfully deployed an anomaly detection model optimized for cloud monitoring, and all research questions have been answered.

Bibliography

[1] Clustering. https://scikit-learn.org/stable/modules/clustering.html. Accessed: 2024-04-04.

[2] Overview of amazon web services. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html. Accessed: 2024-03-13.

[3] sklearn.preprocessing.powertransformer. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html. Accessed: 2024-03-20.

[4] What is amazon cloudwatch? https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html. Accessed: 2024-03-13.

[5] What is amazon sagemaker? https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html. Accessed: 2024-03-13.

[6] What is cloud computing? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing. Accessed: 2024-03-18.

[7] What is cloud computing? https://aws.amazon.com/what-is-cloud-computing/. Accessed: 2024-03-18.

[8] S. Ali, C. Boufaied, D. Bianculli, P. Branco, L. Briand, and N. Aschbacher. An empirical study on log-based anomaly detection using machine learning, 2023.

[9] A. Bifet and R. Gavaldà. *Learning from Time-Changing Data with Adaptive Windowing*, pages 443–448.

[10] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[11] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Serveys*, 41(3), 2009.

[12] X. Chen, W. Xu, S. Wang, Y. Li, and Z. Lin. An anomaly detection scheme with k-means aided extended isolation forest in rss-based wireless positioning system. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1910–1915, 2022.

[13] Z. Ding and M. Fei. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC Proceedings Volumes*, 46(20):12–17, 2013. 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013.

[14] M. Frye, J. Mohren, and R. H. Schmitt. Benchmarking of data preprocessing methods for machine learning-applications in production. *Procedia CIRP*,

104:50–55, 2021. 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0.

[15] S. Guha, N. Mishra, G. Roy, and O. Schrijvers. Robust random cut forest based anomaly detection on streams. In *ICML 2016*, 2016.

[16] S. Hariri, M. C. Kind, and R. J. Brunner. Extended isolation forest. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1479–1489, 2021.

[17] M. Heigl, K. A. Anand, A. Urmann, D. Fiala, M. Schramm, and R. Hable. On the improvement of the isolation forest algorithm for outlier detection with streaming data. *Electronics*, 10(13), 2021.

[18] P. Jain, S. Jain, O. R. Zaïane, and A. Srivastava. Anomaly detection in resource constrained environments with streaming data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(3):649–659, 2022.

[19] E. Khaledian, S. Pandey, P. Kundu, and A. K. Srivastava. Real-time synchrophasor data anomaly detection and classification using isolation forest, kmeans, and loop. *IEEE Transactions on Smart Grid*, 12(3):2378–2388, 2021.

[20] B. Kitchenham and P. Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013.

[21] E. Knauss. Constructive master's thesis work in industry: Guidelines for applying design science research. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 110–121, 2021.

[22] M. T. R. Laskar, J. X. Huang, V. Smetana, C. Stewart, K. Pouw, A. An, S. Chan, and L. Liu. Extending isolation forest for anomaly detection in big data via k-means. *ACM Trans. Cyber-Phys. Syst.*, 5(4), sep 2021.

[23] V.-H. Le and H. Zhang. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 492–504, 2021.

[24] V.-H. Le and H. Zhang. Log-based anomaly detection with deep learning: how far are we? In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1356–1367, New York, NY, USA, 2022. Association for Computing Machinery.

[25] F. Liu. StreamAD, May 2022.

[26] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[27] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab. Machine learning for anomaly detection: A systematic review. *IEEE Access*, 9:78658–78700, 2021.

[28] C. Raab, M. Heusinger, and F.-M. Schleif. Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416:340–351, 2020.

[29] P. Raut, A. Mishra, S. Rao, S. Kawoor, S. Shelke, M. Deore, and V. Kumar. Review on log-based anomaly detection techniques. In S. Shakya, K.-L. Du, and W. Haoxiang, editors, *Proceedings of Second International Conference on Sustainable Expert Systems*, pages 893–906, Singapore, 2022. Springer Nature Singapore.

[30] H. Sun, Q. He, K. Liao, T. Sellis, L. Guo, X. Zhang, J. Shen, and F. Chen. Fast anomaly detection in multiple multi-dimensional data streams. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1218–1223, 2019.

[31] A. Tang, S. Sethumadhavan, and S. J. Stolfo. Unsupervised anomaly-based malware detection using hardware features. In A. Stavrou, H. Bos, and G. Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, pages 109–129, Cham, 2014. Springer International Publishing.

[32] M. U. Togbe, M. Barry, A. Boly, Y. Chabchoub, R. Chiky, J. Montiel, and V.-T. Tran. Anomaly detection for data streams based on isolation forest using scikit-multiflow. In O. Gervasi, B. Murgante, S. Misra, C. Garau, I. Blečić, D. Taniar, B. O. Apduhan, A. M. A. C. Rocha, E. Tarantino, C. M. Torre, and Y. Karaca, editors, *Computational Science and Its Applications – ICCSA 2020*, pages 15–30, Cham, 2020. Springer International Publishing.

[33] M. U. Togbe, Y. Chabchoub, A. Boly, M. Barry, R. Chiky, and M. Bahri. Anomalies detection using isolation in concept-drifting data streams. *Computers*, 10(1), 2021.

[34] V. Werner de Vargas, J. A. Schneider Aranda, R. dos Santos Costa, P. R. da Silva Pereira, and J. L. Victoria Barbosa. Imbalanced data preprocessing techniques for machine learning: a systematic mapping study. *Knowledge and Information Systems*, 65(1):31–57, Jan 2023.

[35] Y. Yang, X. Yang, M. Heidari, M. A. Khan, G. Srivastava, M. R. Khosravi, and L. Qi. Astream: Data-stream-driven scalable anomaly detection with accuracy guarantee in iiot environment. *IEEE Transactions on Network Science and Engineering*, 10(5):3007–3016, 2023.

[36] I. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 12 2000.

A

Questionnaire for Cycle 1

Questions

1. Based on your experience, are the main findings identified in the review applicable to our thesis? (Integrating anomaly detection in AWS)
2. Are there any critical technologies or methodologies that you think could be more appropriate to explore in the thesis?
3. From an implementation perspective, what challenges do you foresee in applying these techniques in AWS for anomaly detection of CloudWatch data?
4. Overall, what are your thoughts about the thoroughness and quality of the literature review?

B

Questionnaire for Cycle 3

Questions for Validating Cycle 3

1. How clear are the guidelines presented to you?
2. Were there any points or terms that were confusing or unclear?
3. Is there any additional information you believe is missing from the guidelines?
4. Do these guidelines adequately cover the steps required for deploying and managing anomaly detection models in AWS?
5. Overall, how useful do you find these guidelines for someone who is new to deploying anomaly detection models in AWS?
6. Would you change the structure or format of these guidelines to make them more user-friendly?