# Monitoring and Maintaining Machine Learning Models Against Concept Drift in the Context of AIOps Systems

Poenaru-Olaru, L.

**DOI**
10.4233/uuid:f660a994-af79-475c-853b-7edc9d1f8b77

**Publication date**
2026

**Document Version**
Final published version

**Citation (APA)**
Poenaru-Olaru, L. (2026). *Monitoring and Maintaining Machine Learning Models Against Concept Drift in the Context of AIOps Systems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:f660a994-af79-475c-853b-7edc9d1f8b77

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

# Monitoring and Maintaining Machine Learning Models Against Concept Drift in the Context of AIOps Systems

Lorena Poenaru-Olaru

# MONITORING AND MAINTAINING MACHINE LEARNING MODELS AGAINST CONCEPT DRIFT IN THE CONTEXT OF AIOPS SYSTEMS

# MONITORING AND MAINTAINING MACHINE LEARNING MODELS AGAINST CONCEPT DRIFT IN THE CONTEXT OF AIOPS SYSTEMS

## Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Wednesday, 21 January 2026 at 17:30 o'clock

by

**Lorena POENARU-OLARU**

Master of Science in Computer Science,
Delft University of Technology, Netherlands
born in Craiova, Romania.

This dissertation has been approved by the promotors.

promotors: Prof. dr. A. van Deursen and Prof. dr. J.S. Rellermeyer
copromotor: dr. L. Miranda da Cruz

Composition of the doctoral committee:

| | |
|---|---|
| Rector Magnificus, | chairperson |
| Prof. dr. A. van Deursen, | Delft University of Technology, promotor |
| Prof. dr. J.S. Rellermeyer, | Leibniz University Hannover, promotor |
| Dr. L. Miranda da Cruz, | Delft University of Technology, copromotor |
| | |
| *Independent members:* | |
| Prof. dr. D. Spinellis, | Delft University of Technology |
| Prof. dr. H. Muccini, | University of L'Aquila, Italy |
| Dr. G.A. Lewis, | Carnegie Mellon University, USA |
| Dr. J. Rubin, | University of British Columbia, Canada |
| Prof. dr. A.E. Zaidman, | Delft University of Technology, reserve member |

The work in the thesis has been carried out within the AI4FinTech Research Lab, a collaboration between ING Bank Netherlands and Delft University of Technology, operating under the Innovation Center for Artificial Intelligence (ICAI) flag.





*Keywords:* AI/ML monitoring, AI/ML maintenance, concept drift, AIOps

*Printed by:* ProefschriftMaken, *https://www.proefschriftmaken.nl*

*Style:* TU Delft House Style, with modifications by Moritz Beller
*https://github.com/Inventitech/phd-thesis-template*

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.



ISBN 978-94-6534-123-1

An electronic version of this dissertation is available at *http://repository.tudelft.nl*.

*Success In Life = The People You Meet + What You Create Together*

Keith Ferrazzi

# Contents

# Summary

The adoption of AI systems across various sectors has increased considerably in recent years. This is a consequence of the remarkable capability of AI to extract insights from large-scale datasets, improve personalization, automate tasks and complex processes within organizations, and support more informed decision-making. Notable examples include the financial sector, where AI is applied to monitor transactions and accelerate credit decision processes; healthcare, where AI contributes to drug discovery and assists clinicians in the early diagnosing; manufacturing, where predictive maintenance using AI systems help reduce costs and mitigate the risks associated with unexpected failures; and software engineering, where AI supports anomaly detection, fault prediction, and resource demand forecasting in large-scale, complex systems.

Despite the widespread adoption and potential of AI systems, most research has been focused on model development, while investigations into their lifecycle and evolution in production environments remain at an early stage. This research path is particularly relevant for AI practitioners, who are responsible for ensuring the reliability, functionality, and predictive accuracy of deployed systems. To bridge the gap between scientific research and the practical needs of industry practitioners, this thesis focuses on two key aspects of the AI lifecycle: techniques for monitoring and maintaining AI systems over time.

We begin this thesis by exploring the accuracy of identifying changes in the data of multiple monitoring techniques, also known as drift detectors (Chapter 2). For the remainder of the thesis, we focus on one particular category of AI systems, namely AIOps systems. Thus, we experiment with the following AIOps system: failure prediction, anomaly detection, and capacity forecasting. Beyond assessing the accuracy of drift detectors, we investigate their potential as indicators for model maintenance, particularly the retraining of failure prediction models (Chapter 3). We further explore whether drift detection-based retraining can enhance the sustainability of such models by reducing energy consumption compared to periodic retraining (Chapter 4). The subsequent chapters extend the analysis to anomaly detection systems (Chapter 5) and examine the practical implications of applying drift detection-based retraining to capacity forecasting models (Chapter 6).

In the final part of this thesis, we summarize our findings and their implications in practice. Our results suggest that drift detection techniques hold significant promise for monitoring AI systems. However, whether they can be indicators for model maintenance is dataset dependent and requires empirical validation. An inappropriate choice of detector may lead to insufficient updates of the AI systems, which can impact the AI system's accuracy or unnecessary retraining. We also observed that, for time series–based systems such as anomaly detection and forecasting, drift detection-based retraining can sometimes result in lower accuracy compared to periodic retraining. This could be due to current limitations of drift detection methods in practical scenarios. Therefore, our findings could serve to highlight the need for further research in the area of drift detection techniques for time series.

# Samenvatting

De adoptie van AI-systemen in diverse sectoren is de afgelopen jaren aanzienlijk toegenomen. Dit is een gevolg van het opmerkelijke vermogen van AI om inzichten te halen uit grootschalige datasets, personalisatie te verbeteren, taken en complexe processen binnen organisaties te automatiseren, en beter onderbouwde besluitvorming te ondersteunen. Opvallende voorbeelden zijn de financiële sector, waar AI wordt toegepast om transacties te monitoren en kredietbeslissingen te versnellen; de gezondheidszorg, waar AI bijdraagt aan medicijnontwikkeling en clinici ondersteunt bij vroege diagnostiek; de industrie, waar voorspellend onderhoud met AI-systemen helpt kosten te verlagen en risico's van onverwachte storingen te beperken; en software-engineering, waar AI helpt bij het detecteren van anomalieën, het voorspellen van fouten en het inschatten van de benodigde middelen in grootschalige, complexe systemen.

Ondanks de brede adoptie en het potentieel van AI-systemen, heeft het meeste onderzoek zich gericht op modelontwikkeling, terwijl studies naar hun levenscyclus en evolutie in productieomgevingen zich nog in een vroeg stadium bevinden. Dit onderzoekspad is vooral relevant voor AI-professionals, die verantwoordelijk zijn voor de betrouwbaarheid, functionaliteit en voorspellende nauwkeurigheid van geïmplementeerde systemen. Om de kloof tussen wetenschappelijk onderzoek en de praktische behoeften van professionals in de industrie te overbruggen, richt dit proefschrift zich op twee kernaspecten van de AI-levenscyclus: technieken voor het monitoren en onderhouden van AI-systemen in de loop van de tijd.

We beginnen dit proefschrift met een verkenning van de nauwkeurigheid waarmee veranderingen in data worden geïdentificeerd door verschillende monitoringtechnieken, ook wel driftdetectoren genoemd (Hoofdstuk 2). Voor de rest van het proefschrift richten we ons op één specifieke categorie AI-systemen, namelijk AIOps-systemen. We doen experimenten met de volgende AIOps-toepassingen: het voorspellen van storingen, het detecteren van afwijkingen en eht voorspellen van capacitieit. Naast het beoordelen van de nauwkeurigheid van driftdetectoren onderzoeken we hun potentieel als indicatoren voor modelonderhoud, in het bijzonder voor het hertrainen van modellen voor storingsvoorspelling (Hoofdstuk 3). Vervolgens onderzoeken we of hertraining op basis van driftdetectie de duurzaamheid van dergelijke modellen kan verbeteren door het energieverbruik te verminderen in vergelijking met periodieke hertraining (Hoofdstuk 4). De daaropvolgende hoofdstukken breiden de analyse uit naar anomaliedetectiesystemen (Hoofdstuk 5) en onderzoeken de praktische implicaties van het toepassen van driftdetectie-gebaseerde hertraining op capaciteitsvoorspellingsmodellen (Hoofdstuk 6).

In het laatste deel van dit proefschrift vatten we onze bevindingen en hun praktische implicaties samen. Onze resultaten suggereren dat driftdetectietechnieken veelbelovend zijn voor het monitoren van AI-systemen. Of ze echter als indicatoren voor modelonderhoud kunnen dienen, is afhankelijk van de dataset en vereist empirische validatie. Een onjuiste keuze van detector kan leiden tot onvoldoende updates van AI-systemen, wat de

nauwkeurigheid van het systeem kan beïnvloeden of tot onnodige hertraining kan leiden. We hebben ook geconstateerd dat bij tijdreeks-gebaseerde systemen zoals anomaliedetectie en voorspellingen, hertraining op basis van driftdetectie soms tot een lagere nauwkerigheid leidt dan periodieke hertraining. Dit kan te wijten zijn aan de huidige beperkingen van driftdetectiemethoden in praktische situaties. Onze bevindingen kunnen daarom bijdragen aan het onderstrepen van de noodzaak voor verder onderzoek naar driftdetectietechnieken voor tijdreeksen.

# Acknowledgments

Looking back, my PhD journey was filled with challenges, growth, and countless memorable moments that shaped me not only professionally but also personally, helping me discover more about myself. Throughout this journey I learned a lot about my strengths and weaknesses, as well as what motivates and inspires me. None of this, however, would have been possible without the incredible people who were part of this journey. I would therefore like to take this opportunity to express my heartfelt gratitude to all of them.

**Luis:** If it weren't for your guidance and support throughout my PhD, I'm not sure I would be here today, preparing for my defense. You have been much more than a supervisor, you have been a mentor with whom I could openly talk about my challenges, feelings, and experiences. You always believed in me and my research, constantly strengthening my confidence in both my work and myself. Your encouragement to take risks, to challenge my limits and to trust my potential has shaped not only my PhD journey, but also my growth as a researcher and as a person. I am deeply grateful to have had a supervisor who always reminded me of how far I have come, especially in moments of self-doubt.

**Arie:** You gave me the opportunity to be part of two incredible research groups that were essential to my PhD journey and personal development, the AI4Fintech Research Group (AFR) and the Software Engineering Research Group (SERG). I have always admired your calm and composed attitude, even in stressful situations, and I am grateful for the example that it set. I also truly appreciate the valuable lessons that I learned from you about increasing my visibility at conferences and proactively creating new opportunities. Despite your busy schedule, you always made time for me whenever I needed guidance or simply wanted to discuss something, whether related to my research or not. I am sincerely grateful to have had you as my promotor.

**Jan:** Thank you for providing me with the opportunity to do this PhD. I have always appreciated your support, positivity, and optimism throughout my PhD, even during paper rejections.

I also would like to express my appreciation to the members of my committee, whom I personally met during my PhD. Thank you for your valuable feedback and for being part of the last step in completing my PhD journey.

**Kim:** You played an important role in ensuring that my PhD goes smoothly. From scheduling meetings with my supervisors to ensuring that all the bureaucratic aspects are properly taken care of, you made sure everything was well handled and I would like to thank you for all these stress-free moments!

To my amazing paranymphs, my lionesses, and true examples of "women empowering women", without whom my PhD journey would never have been the same.

**Eileen:** I am very grateful to have you in my life, not only as my PhD colleague and work bestie, but also as a dear friend. Throughout our PhD journeys, we grew together and helped each other become the best versions of ourselves. When I think back on the

highlights of my PhD years, I immediately remember all the incredible moments we shared, from industry events and conferences to professional galas such as the Women in AI Gala.

**Elvan:** From my very first interview for this PhD position, I admired your professionalism as a manager and your kindness as a person. You are a true leader, always standing by our side and offering your support whenever we needed it. You handled all the processes required to ensure that our PhD experience within the AFR Lab ran smoothly, often without us realizing how much work that entailed. I have learned so much from you and I am truly grateful that my manager has also become my friend.

This PhD thesis would not have been possible without the great people I had the opportunity to collaborate with. From the very beginning, I had one clear goal in mind: I wanted my PhD research to have a real impact in industry. I was able to see the impact of my research in a real-world industry setting thanks to the amazing ING team, *Wouter, Arkadius, Adrian, Evert Jan, and Sergie*, to whom I am deeply grateful for their support, guidance, and the opportunity to work together. It was also a genuine pleasure to supervise *Natalia Karpova* during her thesis and internship at ING, and to witness her impressive growth throughout this important milestone. Last but certainly not least, I would like to thank a special person and friend with whom I had the privilege to work, **June**. You played an important role in my PhD journey, always there to listen to my struggles, offer advice, and brainstorm research ideas together. From the moment we first spoke after your presentation at SERG, I noticed your optimism, humor, and positive energy which made our group an even more welcoming and fun place.

I can call myself lucky enough to be surrounded by so many amazing people throughout my PhD journey. Out of these people I would like to mention my close friends and colleagues. **Shujun:** Your cheerful spirit, the amazing Asian food and bubble tea always lifted my mood. **Sara:** I am very happy you decided to do your MSc thesis in Delft and I am very grateful for your honest and pure friendship. **Taraneh:** Thank you for making the beginning of my PhD memorable and for your long-lasting friendship. **Maliheh:** I deeply appreciated our long discussions after a hard day full of work which helped me unwind, disconnect and relax. *Baris*, *Ege* and *Aru*, I would like to thank you for all the "coffee" and "smoke" breaks we shared which helped me recharge and clear my mind.

During this PhD, I had the privilege of being part of two amazing research groups, AFR and SERG. I would like to thank my colleagues from the AFR lab, *George, Sara, Patrick, Floris, Jerry*, for making my PhD journey such a positive experience, even though it began during the COVID-19 lockdown. I still remember how excited I was at our first group dinner, finally meeting everyone in person. To my SERG colleagues, *Ali Al-Kaswan, Ali Khatami, Amir Deljouyi, Amir Mir, Annibale, Antony, Burcu, Caro, Georgios, Imara, Joao, Jonathan, Mark, Mauricio, Mehdi, Mitchell, Sebastian, Thomas*, I am extremely grateful for all the time we spent together: our (sometimes quite long) coffee chats and lunches, the unforgettable conference trips (especially the ICSE parties, where we brought just the right energy to get things started!), and the wonderful SERG retreats, where we not only learned about each other's research but also truly got to know one another.

I would like to express my sincere gratitude to my Romanian friends: *Iulia, Roxana, Teodora, Oana, Veronica, Livia, Bianca.* You always believed in me, supported me and were there whenever I needed to disconnect. Seeing you, even if not as often as I would have liked, always filled me with positive energy and helped me return to work feeling refreshed

and motivated. I would also like to thank my fluffy friends, *Serena* and *Chuck*, for keeping me company during countless deadlines and (thankfully) not (always) walking across my keyboard and deleting my entire paper drafts.

**Parents + Grandma:** I would like to thank my parents for supporting my ambitions and believing in me and my work. Thank you for standing behind my decision to move to the Netherlands and for encouraging me to embrace new challenges far from my initial home. Your faith in my ability to adapt and overcome all challenges have been a constant source of strength and motivation throughout this journey. You always celebrated my successes so loudly and sincerely that I never even noticed those who did not. None of this would have been possible without your endless love. Aș dori să îi mulțumesc din suflet și bunicii mele pentru sprijinul ei necondiționat, gândurile bune, și credința pe care a avut-o mereu în mine. Încă din anii de liceu glumea pe seama faptului că „lucrez la lucrarea de doctorat" atunci când mă vedea mereu cu laptopul deschis, iar astăzi a sosit cu adevărat momentul să susțin acea lucrare.

**Șerban:** If I had to name my greatest supporter and motivator throughout my PhD, it would undoubtedly be my partner and future husband. You were my rock during the hard times of my PhD, my rational voice helping me see beyond setbacks, and my constant source of encouragement, motivating me to take risks and believe in my ability to shine. You stood by me through my lowest and highest moments, offering unwavering support in the lows and sharing the joy of every success as if it were your own. I am extremely grateful to have you in my life.

*Lorena*
*Delft, January 2026*

# 1

# Introduction

## 1.1 Motivation - Personal Story

This thesis describes my work on monitoring and maintaining ML systems over time, with a particular focus on AIOps systems. It was carried out within the AI for FinTech Research lab (AFR), an industry-academia collaboration between Delft University of Technology and ING Bank. ING Bank is a Dutch multinational bank headquartered in Amsterdam, offering a wide range of financial services worldwide, such as business banking, wholesale banking, etc. My interest in the topic of engineering ML systems, with a particular focus on monitoring and maintaining them once released in production, began in 2019 while I was pursuing my MSc thesis as part of an internship within Exact Netherlands. Exact is a Dutch software company that specializes in accounting and business management solutions for small and medium-sized enterprises. My role was as an intern data scientist and I was part of a large team of talented data scientists and data engineers whose scope was designing data-driven solutions to solve multiple challenges within Exact.

During my internship, I assisted in multiple stages of developing machine learning-based software, from identifying a business problem to gathering data and communicating with business experts, and from building a prototype to deploying machine learning-based software in production. Pursuing this internship as part of my Master's thesis has significantly improved my knowledge of using Artificial Intelligence (AI) and Machine Learning (ML) in real business settings. Furthermore, it has also allowed me to observe the challenges my colleagues were constantly facing as data scientists who develop machine learning-based software that is used to produce business value.

One of the major issues my colleagues faced was the deterioration of the model's prediction accuracy over time. I observed that the deployed ML software often performed with high accuracy for a few months but then experienced a sudden drop in performance. Whenever this occurred, my colleagues had to dedicate significant time to investigating and diagnosing the issue. In most cases, the data the ML model needed to predict had shifted substantially from the training data, either due to natural causes, such as changes in the calculation of a financial indicator, or other factors, such as errors in data extraction from the pipeline. Beyond the time-consuming nature of these investigations, I also noticed

that these data shifts severely impacted stakeholders' perception of the ML software's reliability. As a result, business stakeholders lost trust in the data science team.

When I joined ING as part of my PhD, I had multiple discussions with different data scientists in the organization who were facing the same problem. Most data scientists were questioning whether the changes in the data that could impact the performance of the ML model over time could be monitored in data science applications. However, the research on engineering AI systems from the perspective of their lifecycle was also limited when I began my PhD. For this reason, I decided to address this research gap and help the community of data scientists by researching how to monitor deployed machine learning software and how to maintain machine learning models during their lifecycle to prevent accuracy drops during my PhD.

## 1.2 Background & Context

We begin this section by a short introduction into machine learning systems (Section 1.2.1), followed by an in depth explanation of the phenomenon of machine learning aging (Section 1.2.2). Here we examining the underlying causes of machine learning aging, and presenting how changes in the data impact the performance of machine learning systems. We continue by presenting techniques that have been developed to mitigate the degradation in predictive performance over time of the machine learning systems (Section 1.2.3) as well as techniques used to detect changes in the data that can lead to the degradation (Section 1.2.4). In the following part of this section, we provide a detailed explanation of the AIOps systems, a specific machine learning application for large software systems which we study in this thesis (Section 1.2.5). Finally, we discuss about the GreenAI research field, which has the purpose to encourage designing more sustainable machine learning systems (Section 1.2.6).

### 1.2.1 Introduction in Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that aims to learn patterns from data in order to predict the future or automate processes [1]. The popularity of ML applications grew with the increasing availability of data, which also influenced the adoption of ML techniques in industry settings. Numerous organizations in diverse sectors have embraced these techniques. Out of these sectors we can mention finance [2], agriculture [3], manufacturing [4], healthcare [5] and IT [6–8]. The usage of ML/AI software systems allowed organizations to detect anomalous transactions [2], early identification of diseases in crops [3], predict replacement of different machines used in manufacturing [4], early identification of health problems [5] and identify anomalies and failures in large software systems [7, 8].

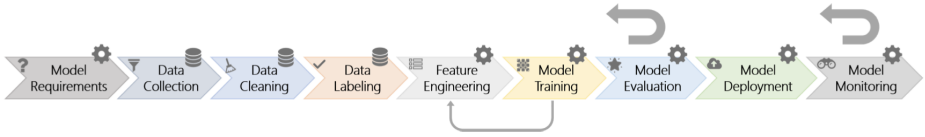



Figure 1.1: Machine learning development workflow according to Amershi et al. [9].

Amershi et al. [9] highlighted a simplified version of the workflow of developing real-world ML application, which we depict in Figure 1.1. According to this workflow, the first step is to define its purpose in terms of *model requirements* and the type of data needed for it as well as to *collect the required data.* Next, the data preprocessing begins, where data need to be *cleaned*, normalized if needed, and *labeled* when developing supervised ML applications. Relevant features have to be derived from preprocessed data as part of *feature engineering.* An ML algorithm is selected based on the problem that the ML application has to address and is used to learn the data from the computed features. During the learning process, also known as *model training*, hyperparameters of the ML algorithms are adjusted accordingly. At the end of the training process, the result is an ML model, which is a term that will be used throughout this thesis. This ML model is further evaluated during the *model evaluation* phase and thereafter deployed in production where *model monitoring* is required.

### 1.2.2 Machine Learning Aging

After deploying ML models into production, the model will continuously receive new data for which it needs to make predictions. However, it has been observed that the predictive quality of AI / ML software systems degrades over time, a phenomenon also referred to as ML/AI aging [10]. This temporal quality degradation of ML/AI software used in real-world application is a critical threat to their reliability and trustworthiness [11, 12].

It has been generally assumed that once an ML system achieves a certain performance after being trained on the available training data, it is prepared for deployment without considering the evolving character of the data that the model is exposed to [10]. However, the ML algorithm used while designing the ML system assumes that the data it has been trained on is similar to the data it will make predictions on. When this assumption does not hold, the predictive quality of the ML system cannot be guaranteed [13]. Thus, the temporal quality degradation of ML/AI systems is usually a consequence of changes in the data over time. These changes in the data over time are also known as concept drift in literature [14]. Unless explicitly mentioned otherwise, in this thesis the term concept drift is used to refer to data changes that impact the model's performance. Concept drift occurs due to uncontrollable factors in real-world data. Examples of concept drift in real-world applications are changes in the financial market which negatively influence the prediction of an ML system designed to predict the credit risk of a person to repay their loan or changes in user behavior and tastes which impact an ML system designed to recommend users content they might like [13]. A real-world example where the impact of concept drift on machine learning was tremendous is Watson for Oncology, in which IBM developed an "Oncology Expert Advisor" system based on AI. This AI system was bought by multiple hospitals and, thereafter, multiple studies about its performance in real-world have been published. Doctors have discovered that the AI system performed poorly in the situation of older patients [1]. Furthermore, medical researchers also discovered that the AI system had a low accuracy on patients with metastatic breast cancer. In this example, the concept drift could have been the data belonging to older patients or patients with metastatic cancer. This concept drift could be a consequence of training AI systems on incomplete data. This

[1] How IBM Watson Overpromised and Underdelivered on AI Health Care: https://spectrum.ieee.org/amp/how-ibm-watson-overpromised-and-underdelivered-on-ai-health-care-2650278241

1

resulted in a high risk of employing such a system in production in the field of medicine and in $62 milion loss for IBM . An example of a more gradual drift is the experience of Zillow with an AI system that could predict real estate prices. The accuracy of the AI system began to drop as housing prices increased. This resulted in massive drops in Zillow shares (around 10%) and a large number of people loosing their jobs [2]. These examples show that data needs to be monitored and that AI systems require regular maintenance through retraining.

### 1.2.3 Adapting Machine Learning Systems to Changes in Data

Through adaptation, we refer to the maintenance required by ML systems to prevent the effects caused by changes in the data over time. This maintenance is usually done through regular model retrainings. When retraining ML systems, we can distinguish two major dimensions: the dimension of the retraining frequency and the dimension of the retraining data. The dimension of the retraining frequency includes two retraining techniques, namely periodic retraining, also referred to as blind adaptation [15] vs. drift detection-based retraining, also referred to as informed adaptation [15]. The dimension of the retraining data is composed of two retraining techniques, namely, the full-history retraining approach and the sliding window retraining approach.

#### Periodic vs. Drift Detection-based Retraining.

Gama et al. [15] distinguishes between two types of retraining techniques for concept drift adaptation, namely *periodic* (blind) vs. *drift detection-based* (informed) retraining. These retraining techniques refer to the frequency of retraining ML systems and their functionalities are depicted in Figure 1.2.

As noticed from Figure 1.2, *periodic retraining* refers to retraining ML systems periodically. The period is usually predefined based on the problem domain and is an engineering requirement of the ML system [16]. On the other hand, the *drift detection-based retraining* approach implies continuous *monitoring* against concept drift using algorithms called *concept drift detectors*. The ML system is retrained only if the concept drift detector identifies changes in the inference data compared to the data the model was trained on. As an example, Figure 1.2 shows a model that is retrained every month based on periodic retraining and only in the month of October because the change in the data has been detected in October based on drift detection-based retraining.

#### Full-History vs. Sliding Window Adaptation

When it comes to the data that is included in the retraining process, Liu et al. [17, 18] have distinguished between two adaptation techniques, namely full-history and sliding window retraining. There retraining techniques refer to the data the ML model is retrained on and the difference between them is shown in Figure 1.3.

From Figure 1.3, we can observe the differences between the full-history and sliding window retraining techniques. The *full-history* retraining approach includes all available data when updating the model. It does not discard old samples, ensuring that past information remains part of the training process. In contrast, the *sliding window* retraining

[2] Zillow to lay off 25% of its workforce and shutter house-flipping service: https://www.cbsnews.com/news/zillow-layoffs-closing-zillow-offers-selling-homes/

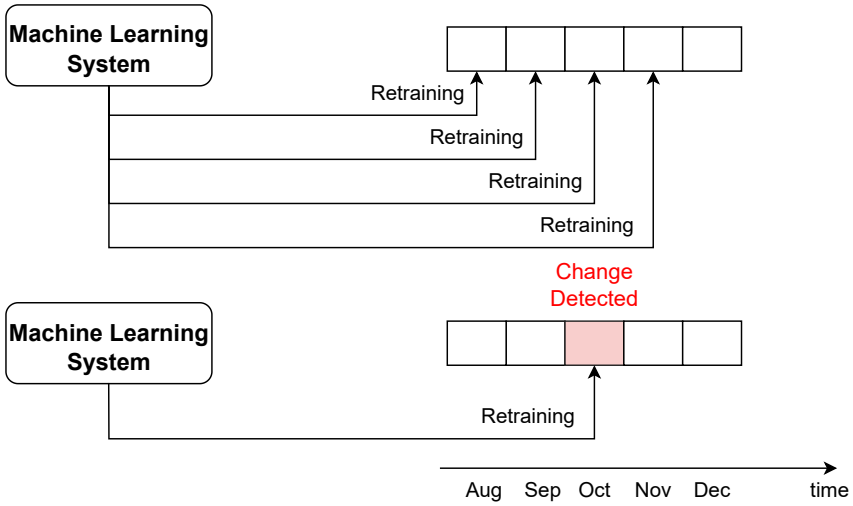



Figure 1.2: Difference between retraining using periodic vs. drift detection-based retraining.

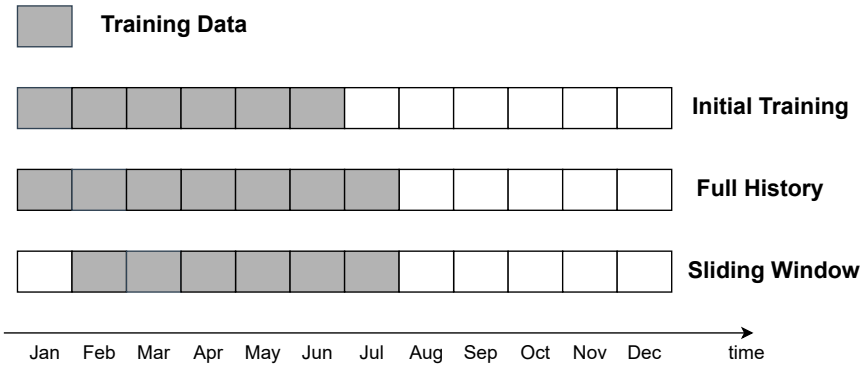



Figure 1.3: Difference between retraining using the full-history and sliding window approach

technique follows a different approach. This method is considered a retraining approach with forgetting mechanisms since it discards old data and retrains the model solely on new samples. It is presented as such in the survey by Gama et al. [15]. Figure 1.3 shows an example in which an ML model is retrained monthly. In the case of full history retraining, the data from January until July is employed to retrain the model, while in the case of the sliding window retraining approach, the data of January is discarded and the model is solely retrained on data from February until July.

### 1.2.4 Concept Drift Detectors Categories for Machine Learning Systems

In their extensive survey, Bayram et al. [14] categorize concept drift detectors based on the specific machine learning problem they are designed for. The authors note that a significant number of detection methods have been developed for classification tasks, largely due to the abundance of open-source datasets available to test these detectors. However, they also highlight that there are significantly fewer concept drift detection techniques for certain areas, such as regression problems or machine learning applications involving input time series data. In this thesis, we investigate two types of machine learning problems, one classification machine learning application, namely failure prediction, and two time series-based machine learning applications represented by capacity forecasting and anomaly detection. Therefore, in our experiments, we include concept drift detectors designed for classification and concept drift detectors designed for time series data.

According to their functionality, concept drift detectors can be divided into error rate-based and data distribution-based drift detectors [19]. The error rate-based detectors, also referred to as *supervised* drift detectors, are drift detection techniques that monitor the performance of an ML system over time. For instance, in classification tasks, these detectors would monitor accuracy when the classes are approximately equally distributed or the ROC AUC score when the classes are highly imbalanced. These detectors are referred to as supervised [14] or label-dependent since they require the correct labels corresponding to the new samples to be available immediately to compute the performance metric and identify drift. On the other hand, the data distribution-based detectors, which are also referred to as unsupervised [14] or label-independent, identify drift solely from the data. Thus, they do not require the true labels of the new samples to be available to identify drift. However, employing supervised drift detection in a real-world setting is impractical since the immediate availability of true labels is not guaranteed or is expensive to obtain [15]. An example where the immediate availability of true labels is impossible is a machine learning system that predicts which customers are going bankrupt within one year. In this situation, the true labels are solely available at the end of the year [15], thus, monthly monitoring data changes using supervised drift detectors is infeasible. An example where gathering true labels is extremely expensive is an application that predicts failures of jobs (processes) in a big complex system [7]. In this situation, gathering true labels to employ supervised concept drift detectors implies that for each failing job, operational engineers investigate the exact cause of the failure and decide whether it was a problem in the system or the reported failure was a human error. For this reason, in real-world ML systems, unsupervised drift detectors are preferred over supervised ones.

### 1.2.5 AIOps

The term AIOps was initially introduced by Gartner and refers to employing AI techniques to improve the efficiency and ensure the quality of large complex software systems [20]. The term AIOps stands for Artificial Intelligence for IT Operations [21], and, recently, plenty of companies are adopting AIOps solutions to manage their IT operations. A recent survey [22] classified AIOps applications into four major categories: *incident detection*, *failure prediction*, *root-cause analysis*, and *automated actions*.

The incident detection category refers to the identification of anomalous behaviors in large-scale complex software systems. Incident detection applications continuously monitor certain predefined system metrics to identify issues related to the system's health, security, or quality. These AIOps applications usually identify anomalies either from time series data [23], such as the CPU utilization over time, or from text data, such as system logs [24]. In this thesis, we dedicate one chapter to anomaly detection AIOps applications.

The failure prediction category contains AIOps applications that could predict problems or incidents in large software systems in advance. These applications monitor different metrics derived from the systems, such as key performance indicator (KPI) metrics or logs. Through the means of advanced algorithms, they identify patterns that may be indicative of potential incidents or failures. This information is then used to notify operational engineers of any system anomalies or misbehavior, enabling proactive intervention and maintenance. Failure prediction algorithms have been used in AIOps applications such as node failure prediction [8, 25], disk failure prediction [6, 26], and job failure prediction [7]. In this thesis, we employ two AIOps failure prediction applications, such as disk failure prediction and job failure prediction.

The root cause analysis category refers to AIOps applications that could indicate possible causes of a specific incident. This category of applications has the potential to reduce the time operational engineers need to spend identifying the root of the problem. An example of root cause analysis AIOps applications is identifying anomalies in parallel in multiple system metrics, such as CPU utilization and network statistics [27].

The automated actions category contains AIOps applications capable of making decisions about the system's properties with the scope of automating its functionality. This category contains both applications that identify the best actions to be taken in a workflow and applications related to resource management according to the demand [22]. In this thesis, we highlight one AIOps application from this category, namely capacity forecasting for resource management.

### 1.2.6 GreenAI

The GreenAI research field focuses on reducing the computational demand of AI systems with the aim of decreasing their energy consumption and enhancing their sustainability [28]. To facilitate the sustainability of ML models, researchers have proposed a series of green tactics that ML/AI systems developers could adopt [29].

It was previously observed by Luccioni et al. [30] that the model training step in the AI/ML model development was the most energy consuming step of the entire process. This step becomes even more energy intensive when the amount of data used for the retraining increases [31]. Given that due to machine learning model aging, AI/ML systems need to be continuously updated due to changes in the data over time [15], we need to

consider not only the energy consumed during the initial model training, but also the energy consumed during each retraining of the model's lifecycle. In this thesis, we provide an empirical study of how different monitoring and maintenance strategies influence the energy consumed during the lifecycle of machine learning systems. One direction that we focus on is investigating whether a concept drift detection-based retraining is able to lower the energy consumption of an ML system lifecycle, given that it can lower the amount of times the model requires retraining.

## 1.3 Research Goal

At the beginning of this research project, we observed that plenty of attention has been paid to assessing the drift detection capabilities of supervised concept drift detectors, while unsupervised drift detectors received less focus. In many real-world applications, unsupervised drift detectors are more suitable to be employed. Thus, we begin this thesis with a comparative study aiming to understand the difference between the two types of drift detectors (M-RQ1) in terms of drift detection accuracy. In the remainder of this thesis, we focused on applications within the AIOps domain. Therefore, the second part of the thesis evaluates whether unsupervised drift detectors can be used to indicate the need for retraining failure prediction systems (M-RQ2). Thus, we initially researched whether unsupervised drift detectors can indicate when the data has changed (M-RQ2.a) and what is the impact of retraining failure prediction models based on drift detection on their energy consumption over time and their accuracy is (M-RQ2.b). In the third part of this thesis, we focus on evaluating unsupervised drift detectors on AIOps applications designed for time series data (M-RQ3). More specifically, we investigate the impact of retraining anomaly detection (M-RQ3.a) and capacity forecasting (M-RQ3.b) AIOps applications based on drift detection.

In this thesis, we aim to answer the following main research questions:

**M-RQ1.** What is the difference in drift detection performance between label-dependent supervised drift detectors and label-independent, unsupervised drift detectors?

**M-RQ2.** What is the impact of retraining failure prediction ML systems based on unsupervised drift detectors?

- **M-RQ2.a.** To what extent can retraining failure prediction ML systems based on unsupervised drift detectors mitigate their degradation in accuracy over time?
- **M-RQ2.b.** To what extent can retraining failure prediction ML systems based on unsupervised drift detectors reduce the energy consumed by this ML application over time?

**M-RQ3.** What is the impact of retraining AIOps applications designed for time series based on drift detection designed for time series data?

- **M-RQ3.a.** What is the impact on anomaly detection accuracy of retraining these ML systems based on drift detection?
- **M-RQ3.a.** What are the practical and performance implications of retraining a capacity forecasting ML system based on drift detection?

## 1.4 Research Methodology

This section contains relevant aspects regarding the research methodology employed to pursue this thesis.

### 1.4.1 Evaluation Framework

Our research aims to investigate whether employing a drift detection-based retraining approach would benefit the ML models over time when it comes to their accuracy, consumed energy, and computational effort. To conduct this research, we require datasets collected over a predefined period for building ML models, existing ML models to investigate the effects of concept drift, and implementations of concept drift detectors for evaluation. For this reason, in this work, we employed multiple datasets for which the timestamp of each sample or group of samples was specified to make our results more generalizable. The ML models that are studied are either replicated/reproduced ML models from literature or proprietary ML models. In terms of concept drift detectors, we used either open-source implementations or we replicated the implementations from literature.

To achieve our goal, we opted for real-world publicly available datasets to ensure that our experiments are relevant to real-world scenarios. However, this imposed more difficulty since industry data is sensitive and not always publicly available. In Chapters 2, 3, 4, and 5, we employed datasets released by organizations for research purpose. Examples of these datasets are the *Backblaze Disk Stats Dataset* released by Backblaze Inc., the *Google Cloud Trace Dataset* released by Google, the *Alibaba GPU Cluster Trace Dataset* released by Alibaba, the *Internet Traffic Yahoo Dataset* released by Yahoo lab, etc. Furthermore, in Chapter 2 synthetic data was also considered to evaluate the precision of drift detectors in identifying the exact moment when drift occurs. Beyond data availability, we also required ML models to evaluate the impact of retraining based on drift detection. For this, we replicated existing ML models (Chapters 3, 4 and 5). In terms of drift detection, implemented drift detectors through replication for Chapters 2, 3, 4, while for Chapter 5 we used an open-source implementation.

Another contribution of this thesis is evaluating this scenario in a real-world industry setting. Thanks to our industry partner, ING Bank Netherlands, we conducted a study on the effect of retraining a real-world AIOps capacity forecasting model based on drift detection (Chapter 6). To pursue this study, we used ING proprietary data related to CPU and Memory utilization and a proprietary ML forecasting model created to improve the capacity management. In terms of drift detectors, we employed the same open-source implementation as for Chapter 5 adapted to the current setting.

### 1.4.2 Code and Data Availability.

Almost all chapters in this thesis rely on open-source data. The source of the open-source data is specified either in the chapter or in its corresponding GitHub repository. To enhance open science and the reproducibility of our results, we publicly share the code employed in each of these chapters. The only exception is Chapter 6, for which we employ proprietary data from our industry partner. For this chapter specifically, the code could not be made publicly available due to privacy reasons. More details about data and code availability are also presented in Table 1.1.

1

Table 1.1: Mapping of research questions to corresponding chapters, along with details on data availability and replication packages. Data is classified as either *public*, indicating that the dataset is open-source or synthetically generated using an open-source framework, or *private*, meaning it consists of proprietary company data. For chapters using public data, the replication packages are openly available, whereas for those relying on private data, the replication package remains private.

| Research Question | Chapter | Data | Replication Package Zenodo |
|---|---|---|---|
| **M-RQ1** | Chapter 2 | public | 10.5281/zenodo.15591540 |
| **M-RQ2** | Chapters 3 and 4 | - | - |
| **M-RQ2.a** | Chapter 3 | public | 10.5281/zenodo.15591505 |
| **M-RQ2.b** | Chapter 4 | public | 10.5281/zenodo.15591579 |
| **M-RQ3** | Chapters 5 and 6 | - | - |
| **M-RQ3.a** | Chapter 5 | public | 10.5281/zenodo.15591569 |
| **M-RQ3.b** | Chapters 6 | private | private |

## 1.5 Research Outline and Findings

In this section, we provide an overview of the chapters of this thesis together with their connection to the research questions. We present further details about the content of each chapter. In Figure 1.4, we present a diagram showing the interconnection between the thesis chapters, as well as a mapping between the thesis chapters and the specific topic covered in each chapter. This thesis explores the impact of concept drift on ML systems, beginning with techniques for detecting concept drift and then examining strategies for adapting ML systems accordingly. We conducted a detailed comparative analysis of various concept drift detection methods to address concept drift detection, evaluating their effectiveness in identifying drift (Chapter 2). Regarding adapting ML systems to concept drift, we further targeted one specific type of ML systems, namely AIOps ML systems. Our choice to focus on this domain was motivated by the fact that AIOps data are usually less sensitive and can be more easily open-sourced or accessed. When studying how to retrain AIOps systems based on concept drift detection, we distinguished between AIOps ML systems designed for multivariate data and those designed for time series data. For AIOps systems handling multivariate data, we focused on failure prediction applications.

First, we examine how drift detection-based retraining impacts the accuracy of failure prediction models (Chapter 3). Then, we extended this analysis to assess the energy consumption of these models over time when retrained based on detected drift (Chapter 4). For AIOps systems designed for time series data, we explored the effects of retraining anomaly detection models based on drift detection on the accuracy of the ML system (Chapter 5). Following this, we investigated the impact of drift-based retraining on a capacity forecasting system (Chapter 6). Since this part of the research was conducted in collaboration with an industry partner, Chapter 6 also examines not only the accuracy implication but also the the industry implications of retraining AIOps ML systems based on concept drift detection.

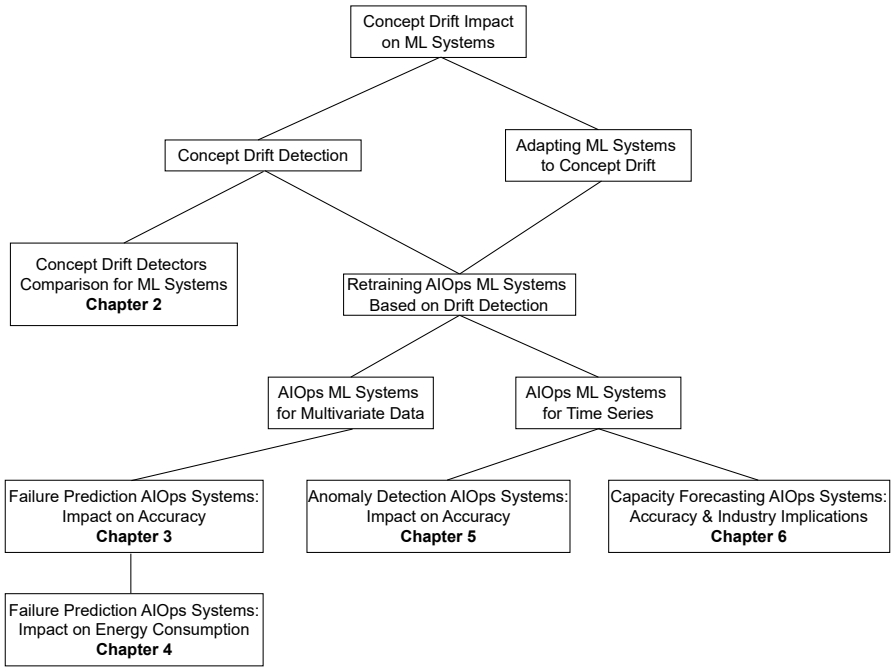



Figure 1.4: Topic and Chapter Mapping.

In Table 1.1, we depict a direct mapping between each research question and its corresponding thesis chapter. Furthermore, this table also gives more details regarding the availability of the data used in each chapter, as well as the replication package. As observed in Table 1.1, this thesis is structured into three main parts: the first part includes Chapter 2, the second part contains Chapter 3 and Chapter 4, and the final part consists of Chapter 5 and Chapter 6.

We begin this thesis with **Chapter 2**, where we focus primarily on machine learning systems for classification purposes that take multivariate data as input. As depicted in Table 1.1, this chapter aims to answer the first research question (RQ1). Previous work [32–34], which studied the performance of supervised drift detectors, has shown that they are accurate in identifying different types of concept drift. However, supervised drift detectors are expensive to employ in real-world classification problems since they rely on the immediate availability of data [15], which makes an unsupervised drift detector a better solution. However, the drift detection accuracy of unsupervised drift detection techniques has not been extensively researched before. Therefore, in this chapter, we assess the drift detection accuracy of an unsupervised drift detector in terms of false alarms, miss-detection rate, and detection latency. This chapter further compares the drift detection accuracy of supervised and unsupervised drift detectors. In doing so, we highlight whether there is a significant research gap between the two categories of drift detectors to identify research gaps that prevent practitioners from employing drift detection

monitoring in real-world ML systems. The comparison is performed employing ML systems developed using both synthetic and real-world data. This study shows that unsupervised drift detection techniques are sensitive to data sparsity and require data scaling, whereas supervised drift detectors remain unaffected by these data characteristics. Consequently, in our following two studies on classification problems (Chapter 3 and Chapter 4), we ensure that the data before using unsupervised drift detectors to identify data changes. Another key finding that resulted from Chapter 2 was the notable discrepancy in the performance of all drift detectors when evaluated on synthetic versus real-world data with high-class imbalance. This observation highlights the need to further investigate ML applications with highly imbalanced classes. As a result, in Chapter 3 and Chapter 4, we evaluate retraining strategies based on drift detection within an ML problem characterized by severe class imbalance.

The next part of the thesis addresses the second research question (RQ2) as shown in Table 1.1. Here, we also focus on studying the effects of retraining ML systems for classification problems using concept drift detection. However, this part, composed of **Chapter 3** and **Chapter 4**, targets a specific ML application from the AIOps research field, namely failure prediction. Failure prediction systems are examples of AIOps applications for which the phenomenon of machine learning aging has been pointed out by previous studies [17, 20, 35, 36]. For this reason, the most commonly known practice to avoid the effects of concept drift on failure prediction models is to periodically update them through retraining [17, 18, 35–37].

**Chapter 3** analyzes the effects of retraining failure prediction models based on drift detection compared to periodic retraining. The novelty of this work lies in the fact that we assess the impact of retraining failure prediction based on drift detection using unsupervised drift detectors, as opposed to supervised methods, which have been previously explored [18]. Obtaining true labels in the case of failure prediction systems is expensive because operational engineers must investigate the root cause of each failure and determine whether it was caused by a real system problem or the result of human error [38]. For this reason, employing supervised drift detectors to monitor whether the model requires retraining is impractical. Therefore, this chapter focuses on identifying which unsupervised drift detection techniques effectively detect when the performance of the failure prediction model begins to degrade. We extract the commonly used unsupervised drift detection techniques mentioned by industry practitioners, namely monitoring the skewness of the features used to build the failure prediction model over time [39] and monitoring the data distribution over time [9, 39, 40] and analyze all using three failure prediction models (one disk failure prediction model and one job failure prediction model). Through this study, we discover that monitoring feature skewness over time is not an effective method to detect failure prediction model degradation. Instead, monitoring the data distribution using distribution-based drift detectors can serve as a useful indicator to detect when to retrain failure prediction models. This approach helps reduce the frequency of retraining while maintaining the accuracy of failure prediction models. However, our findings reveal that the most suitable unsupervised drift detection technique depends on the dataset and must be identified by ML practitioners through experimentation.

**Chapter 4** investigates the effects of retraining failure prediction models based on drift detection from the perspective of the energy consumed over time. Thus, unlike

Chapter 3, which prioritizes the choice of a drift detector that ensures the highest accuracy over time of the failure prediction model, Chapter 4 shows a sustainability vision by evaluating trade-offs between accuracy and consumed energy. Specifically, this chapter quantifies the energy consumed by using various retraining techniques for failure prediction models, with the goal of providing ML practitioners with insight into designing energy-efficient ML systems over time. Details on the accuracy of failure prediction models under different retraining techniques are still included in this chapter to emphasize the influence of the most sustainable retraining approaches on model performance. Although various strategies for designing energy-efficient machine learning systems have been proposed in prior research [29], this work offers a comprehensive evaluation of how different retraining techniques influence the energy consumption of ML systems over their lifecycle. This chapter compares two retraining strategies from the perspective of the retraining frequency [15], retraining periodically vs. retraining based on drift detection. Additionally, it also investigates two other retraining strategies from the perspective of the data that is included in the retraining, namely the full-history approach and the sliding window approach [17, 37]. The full-history approach implies that all the available data is included in the retraining, while the sliding window approach refers to including only the most recent data in the retraining. Our findings reveal that sliding window retraining is more energy-efficient than full-history retraining, while the difference in accuracy between the two approaches remains minimal. Furthermore, we demonstrate that retraining strategies do not affect energy consumption during inference. Lastly, we observe that drift detection-based retraining can reduce energy consumption over time, provided that the drift detector is not overly sensitive to minor fluctuations in data. Based on these findings, we recommend that practitioners adopt a sliding window-based retraining approach in combination with drift detection-based retraining, ensuring that the drift detector is not too sensitive to minor data changes while designing sustainable ML systems.

The last part of this thesis is dedicated to the third research question (RQ3). Here, the focus is no longer on ML systems for classification problems where the data are multivariate, but on ML applications applied to time series data. As aforementioned, the drift detectors applied to time series data also take into account the time dependency of the data, unlike the ones designed for classification problems [14]. This part targets two other AIOps applications, namely anomaly detection, corresponding to **Chapter 5**, and capacity forecasting, corresponding to **Chapter 6**. Although both chapters analyze the same drift detector on two different AIOps applications, Chapter 5 focuses on investigating the effects of retraining multiple anomaly detection techniques based on drift detection and assessing their maintainability over time, while Chapter 6 brings an industry perspective on the implications of retraining based on drift detection from the architectural perspective of the ML system and the current limitations of the concept drift detection techniques.

**Chapter 5** explores various retraining techniques for anomaly detection in AIOps models, aiming to establish maintenance guidelines for ML practitioners responsible for developing such ML applications. Although periodic retraining and concept drift detection-based retraining have been examined in the context of AIOps classification models [17, 18, 37], this study aims to investigate their impact on the accuracy of AIOps anomaly detection models. Throughout this study, multiple anomaly detection algorithms are considered based on their popularity and performance in identifying anomalous samples in AIOps-

related open-source real-world datasets. Similarly to Chapter 4, we include for analysis two retraining techniques related to the retraining data (full-history and sliding window) and two retraining techniques related to the frequency of performing retraining (periodical and concept drift detection-based retraining). Unlike the findings in Chapter 4, which showed no significant difference between using a sliding window and a full-history retraining approach, this study reveals that for anomaly detection, the choice should depend on the employed anomaly detection algorithm. More specifically, we discover that anomaly detection algorithms that take as input the time series in the time domain benefit from a sliding window approach, whereas those that take as input the time series converted into a different domain, such as the frequency domain, should be retrained using a full-history approach. Regarding retraining based on drift detection, our findings indicate that periodic retraining of anomaly detection models achieved slightly higher performance than drift-based retraining. However, since the performance difference between the two approaches was relatively small, we recommend that practitioners adopt drift-based retraining when periodic retraining is computationally expensive or when obtaining true labels is costly.

**Chapter 6** presents a real-world case study together with our industry partner, ING Bank Netherlands, to understand the implications of incorporating drift detection-based retraining into a practical setting. Therefore, the primary contribution of this chapter is a comprehensive evaluation of drift detection-based retraining for an AIOps capacity forecasting model, along with an analysis of its limitations in a real-world context. For this case study, we targeted another AIOps application, namely capacity forecasting, which, similarly to Chapter 5, takes as input time series data. Given our findings in Chapter 5, where retraining based on the Feature Extraction Drift Detector (FEDD) proved beneficial, we selected the same drift detector for this study. Our evaluation aimed to compare whether retraining based on drift detection achieves comparable performance with retraining periodically. The primary motivation for this study was the scalability challenges associated with periodically retraining the capacity forecasting model. Therefore, together with the ING team, we explored the trade-offs in forecasting accuracy between retraining only when the time series is changing versus retraining periodically. If the accuracy implications were minimal, then the team responsible for the forecasting model could opt for a concept drift detection-based retraining to reduce computational costs. Similar to Chapter 5, our results indicated that periodic retraining generally led to higher forecasting performance across most time series. This finding applies especially to situations where the time series exhibited frequent sudden changes. However, unlike the previous study, we also identified scenarios where retraining based on FEDD outperformed periodic retraining, suggesting that incorporating drift detection into the model maintenance pipeline is a promising approach. Moreover, this study provided valuable insights, particularly regarding the limitations of FEDD in practical settings. As a result, we proposed a forecasting system design that integrates FEDD, which addresses some of these limitations, providing a framework that ML practitioners working on time series models can leverage. Additionally, we identified a functional limitation of FEDD in industry applications, which we discuss in this chapter to promote further research on drift detection techniques for time series that can overcome this issue, thereby enhancing the applicability of research in practical industry settings.

## 1.6 Thesis Scientific Contributions

In this section we highlight the main scientific contributions of this thesis.

1. Evaluation of differences in drift detection accuracy between unsupervised and supervised drift detectors for multivariate data across multiple scenarios, showing that the former require more attention from researchers. (Chapter 2)
2. Publicly available implementations of unsupervised drift detectors for multivariate data. (Chapter 2)
3. Evaluation of unsupervised drift indicators such as feature skewness and data distribution monitoring in failure prediction AIOps models in terms of drift detection accuracy. (Chapter 3)
4. Analysis of the impact of retraining AIOps systems based on concept drift detectors as follows:
   (a) Impact on the accuracy for the AIOps systems over time. (Chapter 3, 4, 5, 6)
   (b) Impact on the energy consumption for failure prediction AIOps systems. (Chapter 4)
   (c) Impact on retraining cost savings for capacity forecasting systems. (Chapter 6)

   This analysis is performed on 2 type of AIOps systems, the ones that take as input multivariate data (Chapter 3 and 4) and the ones that take as input time series data (Chapter 5 and 6).
5. Suggestions for machine learning practitioners of which retraining techniques to use to reduce the energy consumption of their machine learning systems over time. (Chapter 4)
6. Replication of state-of-the-art anomaly detection solutions and assessment of anomaly detection retraining techniques. (Chapter 5)
7. Evaluation of the incorporation of a time series drift detector in the retraining pipeline of a real-world capacity forecasting system. (Chapter 6)
8. Highlight of limitations of state-of-the-art drift detectors for time series in a real-world setting and possible setups to overcome them. (Chapter 6)

## 1.7 Thesis Chapters Origins

This dissertation is organized as a collection of research articles. Therefore, some overlap in related work or methodology may occur in different chapters. This is intentional to ensure that each chapter is self-contained when read independently and provides sufficient context for the reader. In this section, we provide an overview of the origins of each chapter. All chapters were peer-reviewed and, for all chapters, the author of this thesis is the first author.

**Chapter 2** is based on the following scientific publication.

1. **Lorena Poenaru-Olaru**, Luis Cruz, Arie van Deursen and Jan S. Rellermeyer. *Are Concept Drift Detectors Reliable Alarming Systems? - A Comparative Study.* IEEE International Conference on Big Data (Big Data), Osaka, Japan, 2022, pp. 3364-3373, https://doi.org/10.1109/BigData55660.2022.10020292. [41]

**Chapter 3** is based on the following scientific publication (publication 3). This chapter was initially presented as a short paper (publication 2).

2. **Lorena Poenaru-Olaru**, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Maintaining and Monitoring AIOps Models Against Concept Drift*. IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN), Melbourne, Australia, 2023, pp. 98-99, https://doi.org/10.1109/CAIN58948.2023.00024. [42]

3. **Lorena Poenaru-Olaru**, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring*, IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest), Ottawa, Canada, 2025, https://doi.org/10.1109/DeepTest66595.2025.00006. [43]

**Chapter 4** is based on the following scientific publication (publication 5). This chapter was initially presented as a short paper (publication 4).

4. **Lorena Poenaru-Olaru**, June Sallou, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Retrain AI Systems Responsibly! Use Sustainable Concept Drift Adaptation Techniques*. IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS), Melbourne, Australia, 2023, pp. 17-18, https://doi.org/10.1109/GREENS59328.2023.00009. [44]

5. **Lorena Poenaru-Olaru**, June Sallou, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Sustainable Machine Learning Retraining: Optimizing Energy Efficiency Without Compromising Accuracy*. 11th International Conference on ICT for Sustainability (ICT4S), Dublin, Ireland, 2025, pp. 100-111, https://doi.org/10.1109/ICT4S68164.2025.00019. [45]

**Chapter 5** is based on the following scientific publication. A summary of this chapter was included in the IEEE Software Practitioner's Digest column [46].

6. **Lorena Poenaru-Olaru**, Natalia Karpova, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Is Your Anomaly Detector Ready for Change? Adapting AIOps Solutions to the Real World*. IEEE/ACM 3rd IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN '24), Lisbon, Portugal, 2024, pp. 222–233, https://doi.org/10.1145/3644815.3644961. [47]

**Chapter 6** is based on the following scientific publication.

7. **Lorena Poenaru-Olaru**, Wouter van 't Hof, Adrian Stańdo, Arkadiusz P. Trawiński, Eileen Kapel, Jan S. Rellermeyer, Luis Cruz and Arie van Deursen. *Prepared for the Unknown: Adapting AIOps Capacity Forecasting Models to Data Changes*. IEEE 36th IEEE International Symposium on Software Reliability Engineering (ISSRE), Sao Paulo, Brazil, 2025. [48]

Included in this thesis.

# 2

# Are Concept Drift Detectors Reliable Alarming Systems? - A Comparative Study

*As machine learning models increasingly replace traditional business logic in the production system, their lifecycle management is becoming a significant concern. Once deployed into production, the machine learning models are constantly evaluated on new streaming data. Given the continuous data flow, shifting data, also known as concept drift, is ubiquitous in such settings. Concept drift usually impacts the performance of machine learning models, thus, identifying the moment when concept drift occurs is required. Concept drift is identified through concept drift detectors. In this work, we assess the reliability of concept drift detectors to identify drift in time by exploring how late are they reporting drifts and how many false alarms are they signaling. We compare the performance of the most popular drift detectors belonging to two different concept drift detector groups, error rate-based detectors and data distribution-based detectors. We assess their performance on both synthetic and real-world data. In the case of synthetic data, we investigate the performance of detectors to identify two types of concept drift, abrupt and gradual. Our findings aim to help practitioners understand which drift detector should be employed in different situations and, to achieve this, we share a list of the most important observations made throughout this study, which can serve as guidelines for practical usage. Furthermore, based on our empirical results, we analyze the suitability of each concept drift detection group to be used as an alarming system.*

## 2.1 Introduction

Predictive algorithms, such as classification algorithms using Machine Learning (ML) on Big Data have seen a significant growth in interest and plenty of real-world applications

This chapter is based on the following peer-reviewed publication:

*Lorena Poenaru-Olaru, Luis Cruz, Arie van Deursen and Jan S. Rellermeyer. 2022. Are Concept Drift Detectors Reliable Alarming Systems? - A Comparative Study. IEEE International Conference on Big Data (Big Data), Osaka, Japan, 2022, pp. 3364-3373* [41].

have been proposed. Examples of those applications are fault detection [49], anomaly detection [50], weather prediction [51], or credit risk prediction [52], where different ML models are constantly evaluated on streaming data. Generally, due to the continuous data flow, data streams are more prone to changes in data distributions over time and, thereby, to concept drift.

Concept drift is a significant threat to the performance of ML models over time. ML models are created by training an ML algorithm on a certain amount of available data, which we are referring to as reference data. The ML algorithms work under the assumption that the data distribution used to evaluate the model is similar to the data distribution of the reference data. However, this assumption does not hold when considering data streams since the evaluation (testing) data is constantly evolving over time due to uncontrollable factors [13]. Therefore, this raises a substantial issue with regards to preserving the performance of ML models over time.

Knowing beforehand when concept drift occurs could help data scientists to take appropriate measures in advance to prevent its effects on the ML model's performance [19]. Thus, special drift algorithms called *concept drift detectors* were proposed to identify the moment when concept drift occurs. They can be used as an alarming system that notifies users about expected model performance degradation. Consequentially, it is important for these drift detectors to be precise when reporting the moment of data shift.

Several studies have identified two major concept drift detectors categories, the *error rate-based* drift detectors and the *data distribution-based* drift detectors [14], [19]. The error rate-based drift detectors identify drift by monitoring the error rate of a trained model on new evaluation data batches. They are always paired with the classification algorithm used to train the model. Since they continuously compute the error rate, these detectors assume that labels are available immediately, which makes them *label-dependent drift detectors*. The data distribution-based drift detectors identify drifts by assessing the similarity between the distribution of the reference data and the evaluation data. There is currently no general similarity metric used uniformly among all studies. Since their drift detection mechanism solely relies on density functions of training and testing data, they are *label-independent drift detectors*. In real-world settings, the data distribution-based detectors are favored over the error rate-based detectors since immediately obtaining labels can be expensive or even impossible [15]. However, recently some techniques were developed to adapt error rate-based detectors for unsupervised and semi-supervised settings [14]. Previous comparative studies [32], [33], [34] focused on analyzing only the error rate-based detectors. Thus, our study is the first to compare the aforementioned two categories of drift detectors. Furthermore, we are the first to assess the precision of detectors in terms of latency and false-positive rate from the perspective of monitoring Big Data ML applications in production and to provide guidelines for practitioners. Thereby, we contribute in the following directions:

1) While previous work [32–34] focuses only on comparing error rate-based drift detectors, in this chapter we assess both the data distribution-based detectors and the error rate-based detectors in terms of false alarms, miss-detection rate and drift detection latency on both synthetic and real-world data.
2) We explore different similarity metrics of data distribution-based detectors and find that, in some cases, other similarity distances are more suitable than the widely used

KL Divergence [53], [54].

3) We share the open source implementation of the data distribution-based detectors employed in this study, which was not previously available. Furthermore, our work is reproducible and available on GitHub.
4) We evaluate the error rate-based detectors paired with three recent and popular classifiers, such as AdaBoost [55], XGBoost [56] and LightGBM [57], as well as commonly used classifiers by previous work [32–34], i.e., Naive Bayes and Hoeffding Trees.
5) We provide some major observations of detector-dataset compatibility, which aim to serve as guidelines for practitioners who want to include drift detectors in their data stream monitoring process.



## 2.2 Background and Related Work

### 2.2.1 Concept Drift General Knowledge

The term concept drift, also known as data shift or data drift, was originally used in data streams to describe changes in data distributions over time [15]. The most common types of concept drift are *abrupt drift* and *gradual drift* [32–34]. The key difference between the two types of drift is the duration. In case of abrupt drift, there is a sudden change in the feature behavior, while in case of gradual change, the features are changing completely after a transition period, as can also be observed in Fig. Figure 2.1. The transition period between the moment when gradual drift starts and the moment it ends is referred to as *drift width*.

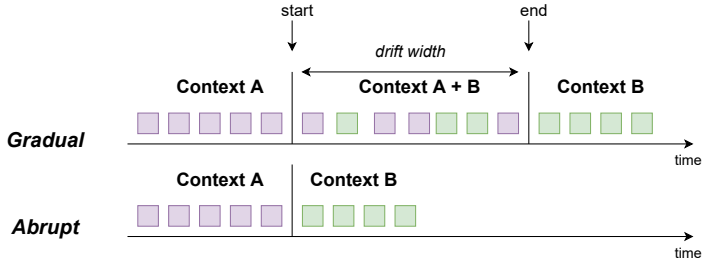



Figure 2.1: Gradual vs. abrupt drift duration.

### 2.2.2 Concept Drift Detectors

Plenty of attention has been paid to developing techniques that are able to detect concept drift as part of data stream monitoring [19]. This section presents drift detectors belonging to both *error rate-based (ERB)* drift detectors and *data distribution-based (DDB)* drift detectors.

Out of the ERB drift detectors, the most popular drift detector is *Drift Detection Method* ***(DDM)*** [58], which uses statistical tests to identify significant changes in error rate. An improved version of DDM is *Early Drift Detection Method* ***(EDDM)*** [59], which, additionally, verifies the distance between error rates when identifying drifts. Another popular ERB drift detector is *Adaptive Windowing* ***(ADWIN)*** [60], a window based technique to store recent

samples. The decrease in mean of the stored samples is monitored to detect drift. The *Drift Detection Method based on the Hoeffding's inequality carried with A-test* ***(HDDM_A)*** *and W-test* ***(HDDM_W)*** [61] methods rely on tracking the moving average and Hoeffding's inequality to determine the significance of the change. Other examples of ERB drift detectors are *FW-DDM* [62], *EWMA chart drift detector* [63] and *RDDM* [64].

Within the (DDB) detectors we distinguish between detectors employing *statistical tests* and detectors using *similarity metrics*. The most popular example of the former category is the *Equal Density Estimation* ***(EDE)*** detector [65], which identifies drift based on a non-parametric statistical tests. The null hypothesis of the tests assumes the similarity of two data distributions and its rejection signals a drift. The most commonly employed DDB detector relying on similarity metrics is *quad-trees which scale with the size (k) and dimensionality (d) of the data* ***(kdqTrees)*** [53]. This technique uses bootstrapping to determine the highest discrepancy between the reference (training) data and subsamples of the reference data in order to compute a critical region. Thereafter, the similarity between the distribution of the new data and the reference data, assessed by the critical region, is used to detect drift. For this technique the similarity metric used is KL Divergence. However, other studies consider different similarity metrics to measure similarities between distributions [66]. Therefore, there is no general similarity metric used in DDB drift detectors and no available study about different metrics suitability in concept drift detection. Furthermore, recent studies suggest that extracting the distributions of the projected features obtained through Principal Component Analysis (PCA) instead of raw features is more suitable for high dimensional datasets [54] and could significantly improve drift detection. Other DDB drift detectors are SyncStream [67] or RD [66].

### 2.2.3 Datasets for Concept Drift Detectors Evaluation

When comparing concept drift detectors, most studies [32–34] are relying on synthetic datasets, usually generated through the MOA Framework [68]. The reason for this is that the moment when the concept drift occurs could be fixed through data generation.

Evaluating concept drift detectors on real-world data is most of the times impractical given that the exact moment when a drift occurs is unknown. However, the study of Webb et al. [69] identifies the moment of drift occurrence for two *real-world datasets*, Electricity (ELECT2) [70] and Airlines [68].

The *ELECT2* datasets, contains samples from Australian New South Wales Electricity Market collected every five minutes over a period of approximately two years. The main prediction problem of ELECT2 is determining whether prices are going up or down based on demand and supply features. In this dataset there is a sudden drift on the 2$^{nd}$ of May when wholesale electricity sales between the Australian Capital Territory, New South Wales, South Australia and Victoria was allowed [69]. The effect of this concept drift could be observed on three attributes of the dataset, which were constant until that date, but started fluctuating afterwards.

The *Airlines dataset*, contains samples corresponding to details of multiple flights collected over a period of four weeks. The main prediction problem is determining whether flights are going to be delayed or on-time. Within this dataset, there is a significant concept drift occurring during the weekend flights (starting from Friday until Sunday) compared to the week days. This drift can be observed especially on the first two weeks of collected

data [69].

## 2.3 Evaluation Methodology



The main goal of this chapter is to evaluate the ability to detect drift in time of both error-rate based (ERB) drift detectors and the data distribution-based (DDB) drift detectors under different conditions. This can be summarized in the following research questions:

**RQ1:** How do state-of-the-art drift detectors compare in their ability to detect abrupt and gradual drift under ideal circumstances?

**RQ2:** How do state-of-the-art drift detectors perform detecting abrupt and gradual drift in the presence of noise and imbalanced data?

**RQ3:** To what extent does the performance of drift detectors on controlled drift position data generalize to real-world data?

### 2.3.1 Data

#### Employed Datasets

In order achieve our goal, we need to know precisely when the drift occurs. Thereby, in our evaluation we include both synthetic data, where the concept drift can be fixed through the data generation process, and real-world data for which we know the moment when the concept drift occurs [69]. Our study exploits three *synthetic datasets*, namely *SEA*, *AGRAW1* and *AGRAW2*, and two real-world datasets for which the moment of concept drift occurrence is known and marked through the findings of Webb et al. [69], namely *Electricity (ELECT2)* and *Airlines*.

We generated synthetic data through two data generators, namely SEA [71] and Agrawal [72] available in the MOA framework. It needs to be mentioned that MOA was solely employed to generate the data, not to perform the evaluation. The former generates three attributes containing numerical features ranging from 0 to 1 and is frequently used in the concept drift detection literature [32, 34], [33]. The latter creates three categorical attributes and six numerical attributes, which correspond to loan-related data. Agrawal generator was created through the process of database mining, in which significant patterns were extracted from large scale industrial data sets and used to generate synthetic data samples. We generated two datasets with the Agrawal generator, *AGRAW1* and *AGRAW2*. Although both AGRAW1 and AGRAW2 were generated using the same generator, they are two different datasets, which consider different forms of evaluation when classifying the samples into the two classes. For all the synthetic datasets, we generated data under ideal conditions, in which no noise was added and the two classes are balanced and also non-ideal conditions, with 10% and 20% noise or imbalanced classes, where the imbalance ratio is 1:2. This is the highest imbalance ratio for which the detectors were able to identify any drift. The scope of the non-ideal conditions is to assess the robustness of the detectors against events that could occurs in real-world scenarios. Furthermore, we generate data for both abrupt and gradual drift. We consider different drift widths, namely [500, 1000, 5000, 10000, 20000] samples. For instance, from the moment the gradual concept drift starts, there are 500 samples until it ends and the features are changing their behavior completely. Each dataset is generated using 10 random seeds to avoid bias in our experiments. We further assess the ability of drift detectors to identify drift on two real-world datasets.

We purposely include datasets containing solely numerical features, SEA and ELECT2, as well as datasets containing both numerical and categorical features, AGRAW1, AGRAW2 and Airlines. In general, categorical features pose problems for ML classifiers, since the ML algorithms usually require numerical values. The most commonly used technique to overcome this issue is one-hot encoding, which converts categorical data into binary vectors. Therefore, we employ this technique in our experiments for the datasets containing a mixture of numerical and categorical features.

After ensuring that all datasets contain solely numerical values, the next step is the data scaling. Two of the datasets considered, SEA and ELECT2, include data scaled between 0 and 1. In order to ensure experimental consistency, we also scale the values for remaining datasets, AGRAW1, AGRAW2 and Airlines. Data scaling was performed using the Min-Max scaler implementation provided in Python scikit-learn version 1.0.2[1]. The reason for choosing this scaler is that it does not make any assumption regarding the distribution of the data following a particular pattern.

### Data Setup

In our experiments we process each dataset as a data stream in which the first part is the reference data and the second part is the testing data. The testing data is divided into equal testing batches. The reason behind this is that the reference data is used to train the ML model which is going to be periodically tested on the new upcoming data. In all cases we ensure that the drift occurs during the testing phase, such that we simulate a deployed ML model which needs to be tested on shifted data. For each new testing batch, a drift detector is employed to determine whether the data has shifted. A detailed representation of our setup is shown in Fig. Figure 2.2. In all our experiments, detectors that signal the drift before the testing batch containing the actual concept drift is a false alarm. In the same manner, signaling the drift after the testing batch containing the drift increases the latency. In case of ERB detectors, the reference data is used in order to train the ML classifiers, which are paired with the concept drift detectors. In case of some DDB detectors, the reference data is used to compute a threshold, which is employed to assess the similarity between the new data and the old data. Furthermore, we solely use the reference data to fit the scaler and then we apply it on each testing batch.

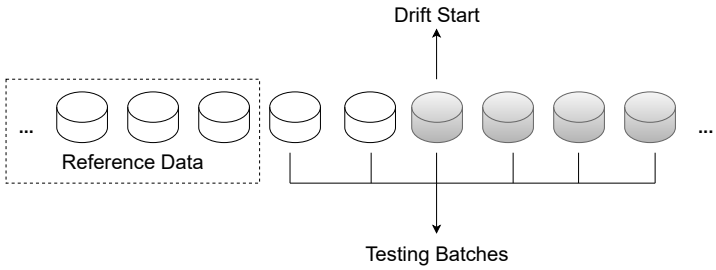



Figure 2.2: Data stream setup.

[1]MinMaxScaler

In the SEA, AGRAW1 and AGRAW2 datasets, the drift start is fixed during the data synthesizing process such that the first two testing batches do not contain drift, while the others include drift.

In case of the real-world data, we initially defined the prediction problem. In case of ELECT2, the prediction problem is weekly predicting whether prices are going up or down. The reference data is composed of the initial part of the data stream, namely data collected between the 07$^{th}$ of May 1996 and the 15$^{th}$ of April 1997. Each testing batch is composed of one week of data. The drift starts in the testing batch containing the 2$^{nd}$ of May 1997. In terms of the Airline dataset, the prediction problem is daily predicting delayed flights. Since the first week of data has missing records corresponding to Monday and Tuesday, we solely consider the second week, for which we have complete data from Monday until Sunday. The reference data is represented by samples from Monday and Tuesday, while the other week days are testing batches. The drift starts on the testing batch corresponding to Friday and lasts until the end of the week.



### 2.3.2 Implementation Decisions in Drift Detectors

When selecting the detectors used for evaluation, our major selection criteria is the publicly available implementations. However, one barrier we encountered was the implementation unavailability of the DDB detectors, for which only mathematical proofs were provided. Thus, we implemented three popular such detectors.

In terms of **ERB** drift detectors we employ DDM, EDDM, ADWIN, HDDM_A and HDDM_W, using the implementations provided in the Python package scikit-multiflow version 0.5.3[2]. These detectors rely on the error rate and, thus, they need to be paired with classifiers. For this study, we use the following classifiers *Naive Bayes*, *Hoeffding Trees*, *AdaBoost*, *XGBoost* and *LightGBM*, which were used either in previous drift detection comparative studies, [34], [32], [33] or in data stream classification [73], [74]. The classifiers are not retrained after a drift is signaled since the purpose of the experiment is solely to identify how fast the first drift can be captured, not to evaluate the situations of multiple drifts. Therefore, the reference data is also not changed after a drift is signaled. For each detector we employed the best hyperparameters configuration.

When it comes to **DDB** drift detectors, we employ the statistical test-based detector EDE and two similarity metric-based detectors, namely kdqTrees and PCA-kdq. We implemented both EDE and kdqTrees according to the original papers [65], [53]. In case of EDE, we employed two non-parametric statistical tests, namely Kolmogorov-Smirnov and Mann Whitney. When it comes to kdqTrees, the original implementation included KL Divergence as similarity metric. However, in our work, we experimented with seven similarity metrics corresponding to seven different groups of distance metrics suitable for determining the similarity between density functions according to Che et al. [75]. Thus, the similarity metrics employed for this study are the following: *KL-Divergence (KL)*, *Manhattan Distance (MH)*, *Chebyshev (CBS)*, *Kulsinski (KLS)*, *Cosine (COS)*, *Squared Euclidean (SE)* and *Bhattacharyya (BTC)*. In terms of PCA-kdq, this detector is a modified version of kdqTrees with the purpose of addressing the high dimensionality. The difference between the two is that instead of extracting the data distribution from the original data, we extract it from the projected data, which is computed through PCA. The similarity metrics employed within

[2] scikit-multiflow

PCA-kdq are the same as the ones used for kdqTrees. All implementations are publicly available in our replication package[3].

### 2.3.3 Evaluation Metrics



To evaluate the drift detectors, we employ three evaluation metrics, namely the *latency*, the *false positive rate* and the *miss-detection probability*. In our study we use these metrics taking into account our data setup with the purpose of understanding how many testing batches with drift are ignored, how many testing batches without drift are signaled as drift and how many datasets with drift are not reported, respectively.

***Latency (L)***: ranges between 0 and 1 and it shows how late the detector manages to detect the drift. If the detector indicates that there is a drift in the first batch when the drift starts, the latency is 0. Therefore, the latency is 0 if the detector identifies the drift at the batch corresponding to the beginning of concept drift in case of gradual drift and occurrence of concept drift for abrupt drift. The formula for the *latency* ($L$) is the following:

$$L = \frac{k-j}{|B|}; b_j, b_k \in B, \tag{2.1}$$

where $b_n$ is the $n^{\text{th}}$ batch in the list of batches ($B$), $b_j$ is the batch corresponding to the beginning of the concept drift, $b_k$ is the batch detected as drift. This metric takes the value ND (nothing detected) if no drift is detected.

***False Positive Rate (FPR)***: shows the percentage of non-drifted batches detected as drifted batches. If no drift is detected in the data-stream, the metric will output ND (nothing detected). The FPR takes the value 0 if no batch that does not contain drift is signaled as drift and 1 if all batches that do not contain drift are signaled as drift. The formula for the *false positive rate* is the following:

$$FPR = \frac{k^F}{|B_{ND}|}; b_k^F \in B, \tag{2.2}$$

where $b_k^F$ is the batch erroneously detected as drift and $B_{ND}$ is the total number of batches without drift out of the total list of batches ($B$).

***Miss-Detection Probability (MDP)***: When evaluating concept drift detectors on synthetic data, it is common to use multiple random seeds of the same dataset to avoid bias. Thus, this metric is only addressed to synthetic data to understand in how many cases the detector managed to identify drift after its occurrence among the 10 random seeds of one dataset, which are referred to as iterations. Since it is a probability, it takes values from 0 to 1, where 0 means that the detector managed to identify drift in all the 10 random seeds of one dataset and 1 means that the detector did not manage to identify any drift in any of the 10 random seeds. The formula for the miss-detection rate is the following:

$$MDP = P(L_{(1,\dots,n)} = ND) \tag{2.3}$$

where n is the number of random seeds, $L_{(1,\dots,n)}$ is the array corresponding to the latency

[3]Replication Package

Table 2.1: Miss detection probability (MDP) of each excluded detector in case of abrupt drift. In case of the ERB detectors we only show the best MDP of each possible configuration of detector+classifier.

| Dataset | Detector Group | Detector | MDP |
|---|---|---|---|
| SEA | ERB | DDM | 1 |
| | | EDDM | 1 |
| | | HDDM_A | 0.8 |
| | DDB | EDE-MW | 1 |
| | | PCA-kdq | 0.8 |
| AGRAW1 | ERB | DDM | 1 |
| | | EDDM | 1 |
| | | HDDM_A | 0.7 |
| | DDB | EDE-MW | 1 |
| | | EDE-KS | 0.8 |
| | | kdqTrees-KL | 1 |
| | | kdqTrees-MH | 0.8 |
| | | kdqTrees-KLS | 1 |
| | | kdqTrees-CBS | 1 |
| | | kdqTrees-COS | 1 |
| | | kdqTrees-SE | 1 |
| | | kdqTrees-BTC | 1 |
| | | PCA-kdq-MHT | 0.8 |
| | | PCA-kdq-CBS | 0.3 |
| | | PCA-kdq-COS | 0.7 |
| | | PCA-kdq-SE | 0.6 |
| AGRAW2 | ERB | DDM | 1 |
| | | EDDM | 1 |
| | | HDDM_A | 0.9 |
| | DDB | EDE-MW | 1 |
| | | EDE-KS | 0.7 |
| | | PCA-kdq-KL | 0.3 |
| | | PCA-kdq-MH | 0.3 |
| | | PCA-kdq-KLS | 0.6 |
| | | PCA-kdq-CBS | 0.3 |
| | | PCA-kdq-COS | 0.3 |
| | | PCA-kdq-SE | 0.3 |
| | | PCA-kdq-BTC | 0.4 |

Acronyms: KL - KL Divergence Distance, MH - Manhattan Distance, KLS - Kulsinski Distance, COS - Cosine Distance, SE - Squared Euclidean Distance, CBS - Chebyshev Distance, BTC - Bhattacharyya Distance, MW - Mann Whitney statistical test, KS - Kolmogorov Smirnov statistical test

## 2.4 Experimental Results

This section presents the performances achieved by error rate-based (ERB) detectors and data distribution-based (DDB) detectors on both synthetic and real-world data.



### 2.4.1 Synthetic Data

**Ideal Conditions.**

With the scope of addressing the first research question, we conduct the first set of our experiments on synthetically generated data under ideal conditions (no noise or class imbalance added).

Table 2.2: Average Latency and FPR of each detector over the 10 iterations for abrupt drift. In **bold** we show the best performing drift detector for each dataset.

| | | | SEA | | AGRAW1 | | AGRAW2 | |
|---|---|---|---|---|---|---|---|---|
| | Detector | | *L* | *FPR* | *L* | *FPR* | *L* | *FPR* |
| ERB | ADWIN | * | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** | **0.00** |
| | HDDM_W | NB | - | - | **0.00** | **0.00** | 0.00 | 1.00 |
| | | HT | - | - | 0.04 | 0.00 | **0.00** | **0.00** |
| | | ADB | - | - | 0.04 | 0.00 | 0.08 | 0.00 |
| | | XGB | - | - | 0.04 | 0.00 | 0.02 | 0.00 |
| | | LGBM | - | - | 0.04 | 0.00 | 0.02 | 0.00 |
| DDB | EDE | KS | **0.00** | **0.10** | - | - | - | - |
| | kdqTrees | KL | 0.00 | 0.20 | - | - | **0.16** | **0.15** |
| | | MH | 0.00 | 0.40 | - | - | **0.04** | **0.30** |
| | | KLS | 0.00 | 0.40 | - | - | **0.12** | **0.20** |
| | | CBS | 0.00 | 0.20 | - | - | **0.12** | **0.20** |
| | | COS | 0.00 | 0.20 | - | - | **0.12** | **0.20** |
| | | SE | 0.00 | 0.15 | - | - | **0.12** | **0.20** |
| | | BTC | **0.00** | **0.10** | - | - | **0.12** | **0.20** |
| | PCA-kdq | KL | - | - | **0.20** | **0.32** | - | - |
| | | MH | 0.00 | 0.25 | - | - | - | - |
| | | KLS | 0.00 | 0.25 | **0.04** | **0.41** | - | - |
| | | CBS | 0.00 | 0.30 | - | - | - | - |
| | | COS | 0.00 | 0.25 | - | - | - | - |
| | | SE | 0.00 | 0.25 | - | - | - | - |
| | | BTC | 0.00 | 0.30 | **0.07** | **0.36** | - | - |

Acronyms: NB- Naive Bayes, HT - Hoeffding Trees, ADB - AdaBoost, XGB - XGBoost, LGBM - LightGBM, KL - KL Divergence Distance, MH - Manhattan Distance, KLS - Kulsinski Distance, COS - Cosine Distance, SE - Squared Euclidean Distance, CBS - Chebyshev Distance, BTC - Bhattacharyya Distance, KS - Kolmogorov Smirnov statistical test

We begin our evaluation by assessing the MDP of each detector on each synthetic dataset in case of abrupt drift. Given that all evaluated datasets contain concept drift injected in the process of data generation, we consider that not being able to flag a drift in one iteration of a dataset is an exclusion criteria for the drift detectors in further experiments.

Thus, we filter out all detectors with a MDP higher than 0.0 for each dataset. We provide a detailed explanation into which detectors are removed during this step for each dataset together with their corresponding MDP in Table 2.1. We observe that a high number of DDB detectors achieve an MDP close to 1 in case of AGRAW1 and AGRAW2 datasets, where the categorical data was encoded using one-hot encoding. This shows that the detectors are unable to find differences between the reference data and the upcoming testing data, which could be a consequence of computing the data distribution from a sparse dataset.



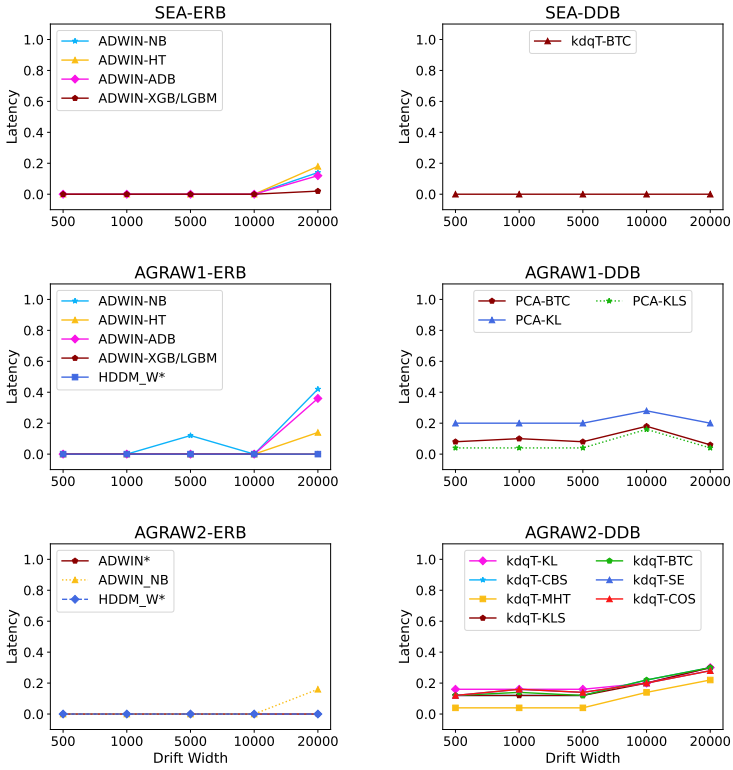



Figure 2.3: Latency of the best performing detectors on different gradual drift width. Each row corresponds to one dataset, SEA, AGRAW1 and AGRAW2. Each column corresponds to the drift detectors type, ERB and DDB.

We continue our experiments by assessing the latency and false positive rate of the remaining detectors on each dataset with abrupt drift. In Table 2.2 we show the results of our findings. The main observation that we can draw from Table 2.2 is that the error-rate based (ERB) detector ADWIN achieves the lowest latency and false positive rate on all datasets, managing to correctly identify all drifts. Furthermore, its performance is

independent of the chosen classifier. When it comes to DDB detectors, we can see that they are in general less precise than the ERB detector ADWIN. Furthermore, there is no general similarity metric or statistical test that achieved the highest performance for all datasets. For both datasets AGRAW1 and AGRAW2, there is no best option in terms of choosing one drift detector, since in all cases there is a compromise between latency and false positive rate. For instance, while KL Divergence minimizes the false positive rate, the Kulsinski and Bhattacharyya distance minimize the latency. Moreover, the DDB detectors can more accurately identify concept drift within the dataset SEA, compared to the datasets AGRAW1 and AGRAW2 datasets.

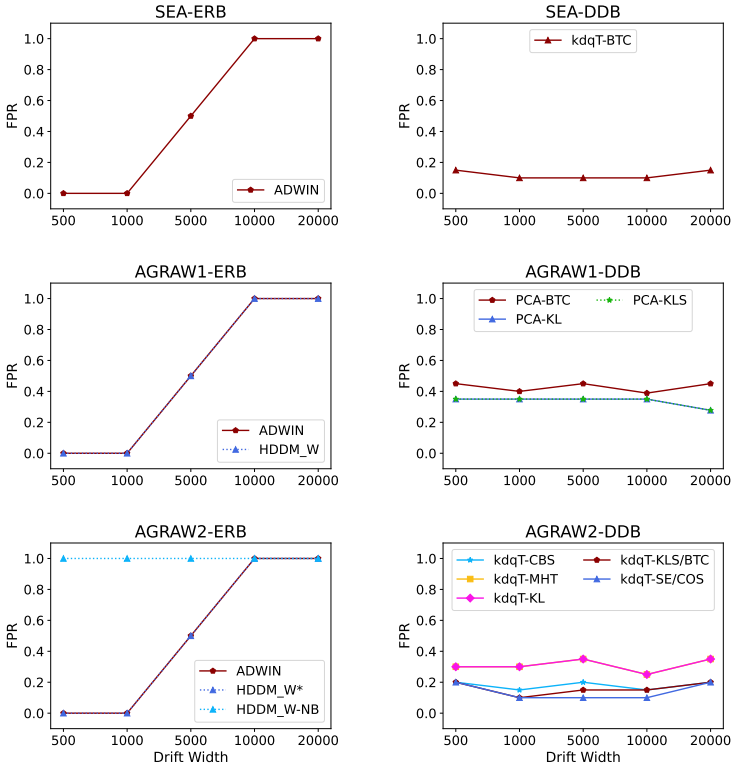



Figure 2.4: False positive rate of the best performing detectors on different gradual drift width. Each row corresponds to one dataset, SEA, AGRAW1 and AGRAW2. Each column corresponds to the drift detectors type, ERB and DDB.

We further assess the precision of both ERB and DDB detectors to identify gradual drift. In this experiment, we solely include the drift best performing drift detectors from

the abrupt drift experiment. The reason for this decision is that in real-world settings the type of drift that might occur is unknown and, thereby, we need detectors which work well on both abrupt and gradual drift. We depict the latency and false positive rate of the chosen detectors in Figure 2.3 and Figure 2.4, respectively.

One observation that we can make from Fig. Figure 2.3 is that the latency in general not impacted by drift widths lower than 10000 samples. We can notice that the latency of ERB detectors increases slightly for a gradual drift width of 20000 samples. Furthermore, the latency of DDB detectors is overall higher than the latency achieved by the ERB detectors.

From Figure 2.4, we can see that the ERB detectors are severely impacted by higher drift widths, with the false positive rate increasing up to 1.0 for 10000 and 20000 samples. However, the false positive rate of DDB detectors remains relatively stable across the different evaluated drift widths.

#### Non-Ideal Conditions

To answer the second research question, we conduct experiments on both abrupt and gradual drift under non-ideal conditions, such as noisy data and class imbalance. In terms of the gradual drift we fix the drift width to 1000 samples, since we noticed from the previous experiment that this is the higher evaluated drift width for which the false positive rate remains 0.0 in case of ERB detectors. However, we observed that the performance of identifying drift in time of ERB and DDB is not affected by the presence of noise. Thus we are not reporting results from this experiment in this section, but we are arguing about the results in the discussions Section. In this experiment we consider the same detectors evaluated on the gradual drift.

**Class-Imbalance:** One notable outcome of this experiment is the inability of DDB detectors to identify any concept drift when the two classes are imbalanced. This is supported by the increase in miss detection rate, which can be observed in Table 2.3. It needs to be mentioned that, similarly to the gradual drift experiment, we solely considered detectors that obtained a miss detection rate equal to 0.0 during the abrupt drift experiment. We can remark from Table 2.3 that the miss detection rate increases for all detectors, except for the kdqTrees paired with the Manhattan distance when evaluated on the dataset SEA with abrupt drift. However, on the dataset SEA with gradual drift, we can still observe a 0.2 increase of the miss detection rate, showing that this detector was not able to detect any drift in 2 out of 10 random seeds.

Since we are dealing with class imbalance, during the experiments with the ERB detectors we initially applied SMOTE [76], which is a commonly used technique that synthetically generates synthetic samples of the minority class. The reason behind this decision is that the classifiers that are paired with the detectors need balanced data to properly learn the behavior of the samples belonging to the two classes. SMOTE was solely applied on the training data in the process of training the classifiers.

In Table 2.4 we show the performances of the two ERB detectors on imbalanced classes. Despite achieving a latency of 0.0, we can notice that the FPR of the HDDM_W detector significant increased in this setup, signaling every testing batch as a drift batch. Thus, this detector tends to signal a high number of false alarms when used in a real-world setting. When it comes to ADWIN, we can notice that its latency significantly increased compared to the case when the two classes are balanced presented above, but the FPR remains constant at 0.

Table 2.3: Miss detection probability (MDP) of DDB detector on class imbalance experiment.

| Dataset | Detector | MDP Abrupt | MDP Gradual |
|---|---|---|---|
| SEA | kdqTrees-KL | 0.6 | 0.7 |
| | kdqTrees-MH | 0.0 | 0.2 |
| | kdqTrees-KLS | 0.9 | 0.9 |
| | kdqTrees-CBS | 0.3 | 0.1 |
| | kdqTrees-COS | 0.3 | 0.3 |
| | kdqTrees-SE | 0.3 | 0.4 |
| | kdqTrees-BTC | 0.9 | 0.9 |
| | PCA-MH | 0.2 | 0.2 |
| | PCA-KLS | 0.5 | 0.5 |
| | PCA-CBS | 0.2 | 0.2 |
| | PCA-COS | 0.2 | 0.2 |
| | PCA-SE | 0.2 | 0.2 |
| | PCA-BTC | 0.7 | 0.6 |
| AGRAW1 | PCA-KL | 0.5 | 0.5 |
| | PCA-KLS | 0.5 | 0.6 |
| | PCA-BTC | 0.6 | 0.6 |
| AGRAW2 | kdqTrees-KL | 0.8 | 0.7 |
| | kdqTrees-MH | 0.6 | 0.7 |
| | kdqTrees-KLS | 0.7 | 0.7 |
| | kdqTrees-CBS | 0.7 | 0.7 |
| | kdqTrees-COS | 0.7 | 0.7 |
| | kdqTrees-SE | 0.7 | 0.7 |
| | kdqTrees-BTC | 0.7 | 0.7 |

Acronyms: KL - KL Divergence Distance, MH - Manhattan Distance, KLS - Kulsinski Distance, COS - Cosine Distance, SE - Squared Euclidean Distance, CBS - Chebyshev Distance, BTC - Bhattacharyya Distance

### 2.4.2 Real-World Data

This last set of experiments seek to answer RQ3, by understanding how do the analyzed drift detectors perform on real-world data. Here we do not know whether the observed concept drift is abrupt or gradual, but only the position of the drift occurrence.

#### Electricity (ELECT2)

As aforementioned, we assessed the detectors' performances to detect the week in which the 2$^{nd}$ of May 1997 is included and we show the results in Table 2.5. Here we notice that both ERB and DDB detectors succeed in identifying the exact testing batch which contains the drift. Specifically, the ERB detector called ADWIN managed to exactly identify the drifted batch, independently of the paired classifier. Furthermore, the same results were reported for the DDM classifier paired with Naive Bayes, Hoeffding Trees and AdaBoost. We can further see that using DDM with XGBoost or LightGBM significantly increases its false positive rate from 0.0 to 1.0, enhancing the risk of false alarms. On the other hand, comparable performance was obtained by one DDB detector, namely PCA-kdq using the

Table 2.4: Latency (L) and False Positive Rate (FPR) for Error Rate-Based detectors on balanced vs. imbalanced data. * shows that the results are applicable to all paired classifiers except for the ones presented. - shows that the experiment is not applicable.

| | Detector | Paired with | SEA | | AGRAW1 | | AGRAW2 | |
|---|---|---|---|---|---|---|---|---|
| | | | *L* | *FPR* | *L* | *FPR* | *L* | *FPR* |
| Abrupt | **ADWIN** | * | 0.8 | 0.0 | 0.8 | 0.0 | 0.8 | 0.0 |
| | | ADB | - | - | - | - | 0.72 | 0.0 |
| | | HT | - | - | - | - | 0.82 | 0.0 |
| | **HDDM_W** | * | - | - | 0.0 | 1.0 | 0.0 | 1.0 |
| Gradual | **ADWIN** | HT | 0.72 | 0.0 | 0.72 | 0.0 | 0.82 | 0.0 |
| | **ADWIN** | * | 0.8 | 0.0 | 0.8 | 0.0 | 0.8 | 0.0 |
| | **HDDM_W** | * | - | - | 0.0 | 1.0 | 0.0 | 1.0 |

Acronyms: HT - Hoeffding Trees, ADB - AdaBoost

Kulsinski distance, which also managed to obtain both latency and false positive rate of 0.0. Thereby, for this real-world dataset, DDB detectors managed to achieve comparable good results with ERB detectors.

Table 2.5: Latency (L) and False Positive Rate (FPR) of each detector on ELECT2 dataset. Each detector is paired with either a classifier (for ERB) or a distance/statistical test (for DDB). In **bold** we show the best performing drift detector(s) from each group. * shows that the results are applicable for any combination and *- shows that results are applicable for any combination except the presented one.

| Group | Detector | Paired with | *L* | *FPR* |
|---|---|---|---|---|
| ERB | **DDM** | **NB, HT, AB** | **0.0** | **0.0** |
| | DDM | XGB, LGBM | 0.0 | 1.0 |
| | EDDM | * | 0.0 | 1.0 |
| | **ADWIN** | * | **0.0** | **0.0** |
| | HDDM_W | * | 0.0 | 1.0 |
| | HDDM_A | * | 0.0 | 1.0 |
| DDB | EDE | * | 0.0 | 1.0 |
| | kdqTrees | KLS | ND | ND |
| | | *- | 0.0 | 1.0 |
| | **PCA-kdq** | **KLS** | **0.0** | **0.0** |
| | | *- | 0.0 | 1.0 |

Acronyms: NB - Naive Bayes, HT - Hoeffding Trees, AB - AdaBoost, XGB - XGBoost, LGBM - LightGBM, KLS - Kulsinski Distance

#### Airlines

As previously mentioned, in case of this dataset the detectors should detect drift on the evaluation batch corresponding to Friday. In Table 2.6 we depict the results for both ERB

Table 2.6: Latency (L) and False Positive Rate (FPR) of each Error Rate-Based (ERB) detector on Airlines dataset. In **bold** we show the best compromise between the latency and false positive rate. * shows that the results are applicable for any combination and *- shows that results are applicable for any combination except the presented one.



| Group | Detector | Paired with | *L* | *FPR* |
|---|---|---|---|---|
| ERB | DDM | NB, HT | ND | 0.5 |
| | | *- | 0.0 | 1.0 |
| | EDDM | NB | ND | 0.5 |
| | | HT, AB, LGBM | 0.0 | 1.0 |
| | | XGB | 0.67 | 1.0 |
| | **ADWIN** | **NB** | **0.0** | **0.5** |
| | | *- | 0.0 | 1.0 |
| | HDDM_W | * | 0.0 | 1.0 |
| | HDDM_A | * | 0.0 | 1.0 |
| DDB | EDE | * | 0.0 | 1.0 |
| | kdqTree | KLS | ND | ND |
| | kdqTree | *- | 0.0 | 1.0 |
| | PCA-kdq | KLS | ND | ND |
| | PCA-kdq | *- | 0.0 | 1.0 |

Acronyms: NB - Naive Bayes, HT - Hoeffding Trees, ADB - AdaBoost, XGB - XGBoost, LGBM - LightGBM, KLS - Kulsinski Distance

detectors and DDB detectors. Here we can observe that the ERB detectors show a poor performance on the Airlines datasets in terms of latency and false positive rate. Most of the detectors capture the exact moment of drift occurrence, but with the high cost of signaling false alarms. The best false positive rate (0.5) and latency (0.0) was reported by ADWIN paired with the Naive Bayes classifier. The high number of false positives can also be observed in case of most DDB detectors, except for kdqTrees and PCA-kdq paired with the Kulsinski distance, where they did not manage to identify any drift. Thus, both ERB and DDB detectors are affected by false alarms.

## 2.5 Discussions

This section highlights the most important observation that we made during our study regarding the two groups of concept drift detectors, namely the *error rate-based (ERB)* detectors and the *data distribution-based (DDB)* detectors. Therefore, we aim to help practitioners employ the most suitable drift detector according to their data.

**ERB detectors proved to be more suitable for datasets including both categorical and numerical features compared to DDB detectors** One major observation that we can draw from our experiments addressing RQ1 and RQ3 is the fact that DDB detectors achieve higher performance on datasets with solely numerical values, such as SEA and ELECT2, compared to datasets with both numerical and categorical values, such as

AGRAW1, AGRAW2 and Airlines. This could be a consequence of the one-hot encoding technique used to transform categorical variables into numerical. This preprocessing technique increases the sparsity of the dataset, since it represents each categorical value as a binary vector. Sparsity usually alters the representation of the data distribution given that the density function is computed using a high number of 0s and 1s [77]. This impacts the performance of DDB detectors due to their high dependency on data distributions. However, the ERB detectors do not suffer from this problem, since they rely on the performance of the classifiers, which are robust towards sparse data.



**DDB detectors can achieve high performance solely when the data is scaled** During our experiments we scaled all datasets, such that their values would range in the interval of [0, 1]. Although scaling is a common practice in ML, it is not necessary when using tree-based algorithms, since they are already robust to widely distributed data [78]. Therefore, we observed that data scaling did not impact the ERB detectors, which rely on ML classifiers' performances. However, we noticed a high impact of unscaled data on the performance of DDB detectors, which were not able to identify any drift. This could be the result of the fact that they solely rely on the data distribution to detect drifts. Having values widely distributed results in a skewed density function, which impacts the ability of similarity metrics to identify significant discrepancies between two data distributions. Furthermore, we experimented with different scaling intervals, but the [0, 1] interval was the most suitable for all the analyzed datasets.

**ERB detectors outperform DDB detectors for abrupt and gradual drift with a small drift width, but suffer from a high number of false alarms in case of gradual drift with a large drift width** When conducting experiments for RQ1, we empirically proved that in case of abrupt and small width gradual drift, the ERB drift detectors outperform the DDB drift detectors, achieving a lower latency and a lower false positive rate. The best performing ERB drift detector overall is ADWIN, which obtained the best latency and false positive rate independently of the chosen dataset or the paired classifier. However, when tested on synthetic data which contains gradual drift with large drift width, the ERB detectors starts signaling multiple false alarms, although the latency is not affected. The same behavior can be noticed when testing the ERB drift detectors on the Airlines dataset, where all detectors suffer from a significantly high false positive rate, which can indicate that this real-world dataset contains a gradual drift. In real-world data the drift type, abrupt or gradual, cannot be controlled. Thus, in a real-world scenarios we should use a detector that is able to identify all types of drifts. Consequently, it is doubtful whether ERB detectors could be employed in practice.

**Given the high discrepancy between synthetic and real-world data, there is currently no clear evidence regarding the fact that class imbalance influences the impact of either DDB or ERB detectors** We investigated the effect of class imbalance on concept drift detectors. In case of synthetic data, we noticed that all evaluated detectors suffer from sever performance degradation, even for a small class imbalance ratio of 1:2. However, on the real-world data the detectors behavior was completely different. On the ELECT2 dataset, both ERB and DDB detectors managed to accurately identify concept

drift even if the imbalance ratio of the drifted batch was approximately 1:6. However, on the Airlines dataset, both ERB and DDB detectors encountered difficulties when detecting the concept drift in time, although the imbalance ratio of the drifted testing batch was smaller than the one on the synthetic data, namely 1:1.66. This casts doubt on whether the synthetic data manages to mimic the behavior of real-world data when it comes to class imbalanced datasets with concept drift and shows how the performance of detectors on controlled drift position does not generalize to real-world data (RQ3). Therefore, there is no clear evidence of how the class imbalance influences the performance of drift detectors.

**When using DDB detectors in practice, multiple similarity distance should be evaluated and in some cases a compromise between latency and false positive rate is required** Another observation that we want to highlight is regarding the DDB detectors. In literature, the most commonly used similarity metric in this detector category is KL Divergence. However, in our experiments for RQ1 and RQ3 we noticed that this similarity metric did not always achieve the lowest latency or false positive rate. When it comes to the synthetic data, the KL Divergence is mostly minimizing the false positive rate, while the Kulsinski Distance or the Bhattacharyya distance minimized the latency. Thereby, when employed in practice, for some datasets a compromise should be made regarding whether the latency should be prioritized over the false positive rate or vice-versa. Furthermore, on the ELECT2 real-world dataset, the PCA-kdq detector paired with the Kulsinski distance achieved the lowest latency and false positive rate. Furthermore, we have not identified any optimal configuration of drift detector + similarity metric that achieved the best performance on all datasets.

**The presence of noise does not impact the latency or false positive rate of either ERB or DDB detectors on synthetic data** When answering RQ2, we noticed that the latency and false positive rate of both ERB and DDB detectors are relatively stable against noise. The explanation behind this aspect is that both the reference data and the evaluation data are affected by the same type and percentage of noise. Thus, the differences between the reference data and evaluation data are too small to impact the performance of evaluated drift detectors. Unfortunately, the MOA framework does not have an option to select which parts should be affected by noise or to include different noise percentages in different parts of the data stream. Therefore, we could not investigate the effect of having clean reference data and noisy evaluation data.

## 2.6 Conclusions and Future Work

In this chapter we have provided an in depth comparison between two categories of drift detectors, the error rate-based drift detectors and the data distribution-based drift detectors under different conditions, synthetic data with ideal conditions, synthetic data with non-ideal conditions and real-world data. For the latter, we have explored multiple similarity metrics and we have observed that some similarity metrics achieved better latency and false positive rate compared to the state-of-the-art KL Divergence on some datasets. Furthermore, we implemented the most popular data distribution based drift detectors and publicly shared them on GitHub and we evaluated the error rate-based drift

detectors on three recent and popular classifiers. Additionally, we provided a list of major observations, which aim to serve as guidelines for practitioners that want to include drift detectors to monitor streaming data.

Our observations indicate that the analyzed concept drift detectors are not fully reliable when used as alarming systems. We show empirical evidence of the fact that the error-based drift detectors are signaling false alarms for a high drift width. This questions their ability to detect drifts in environments where the features are slowly changing over time. An example of such situation is the inflation, which does not have immediate impact on the financial features, but it affects them over a longer period of time. When it comes to data distribution-based drift detectors, their performance overall was lower than the error-based drift detectors, although they were not affected by higher drift widths. We observed that in cases of datasets with categorical features, the data distribution-based detectors suffered from high false positive rate. Furthermore, they are also affected by a high miss-detection rate, which can be a consequence of computing the data distribution from a sparse dataset resulted after applying one-hot encoding. This reduces the reliability of drift detectors when used as alarming systems.



**Future Work:** In order to advance the field of concept drift detection, we believe that research should focus on developing more data distribution-based detectors. One major limitation of these detectors was the high false positive rate. This might be an indicator that they are too sensitive to small changes in data. However, these small changes might not affect the performance of the ML models. Thus, a promising research direction is exploring which similarity metrics are the least impacted by small changes in data. Moreover, changes in some features might affect the ML models' performances than others. Thereby, another way to improve these drift detectors is to identify the most significant features and monitor drift by computing the data distribution corresponding to only those features. Furthermore, we can explore other ways of encoding categorical data that can reduce the data sparsity, since we noticed that their performance was much lower on datasets where one-hot encoding was employed. When it comes to error rate-based detectors, future research should focus on adapting them to identify gradual drift with a large drift width without signaling false alarms and, thereby, preserving the false positive rate. This study was limited to publicly available implementations of drift detectors. Thereby, we strongly encourage researchers who implement new drift detectors to publicly share their code. Furthermore, in the situation of class imbalance we noticed a strong inconsistency between synthetic and real-world data when it comes to the performance of error rate-based detectors and data distribution-based detectors. Thus, the concept drift research path would benefit from understanding whether the synthetic data is suitable to simulate the behavior of real-world data in case of highly imbalanced classes.

## Acknowledgment

This work was partially supported by ING through the AI for Fintech Research Lab with TU Delft.

# 3



# Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring

*Failure prediction models can be significantly beneficial for managing large-scale complex software systems, but their trustworthiness is severely affected by changes in the data over time, also known as concept drift. Thus, monitoring these models against concept drift and retraining them when the data changes becomes crucial in designing reliable failure prediction models. In this work, we evaluate the effects of monitoring failure prediction models over time using label-independent (unsupervised) drift detectors. We show that retraining based on unsupervised drift detectors instead of periodically reduces the cost of acquiring true labels without compromising accuracy. Furthermore, we propose a novel feature reduction for unsupervised drift detectors and an evaluation pipeline that practitioners can employ to select the most suitable unsupervised drift detector for their application.*

## 3.1 Introduction

Failures in large-scale software systems generate service interruptions [20] and are challenging to manage due to their complexity [20]. Consequently, plenty of attention has been directed toward automated techniques that use Artificial Intelligence (AI) and machine learning (ML) to detect failures (AIOps) [20], which are also referred to as failure prediction models. It has been observed that failure prediction models are efficient solutions

This chapter is based on the following peer-reviewed publication:

*Lorena Poenaru-Olaru, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. 2022. Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring. Proceedings of the 6th IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning, Ottawa, Canada, 2025* [43].

for handling system issues which can detect around 70% of a system's failure within 10 minutes [38].

Although beneficial, failure prediction models suffer from temporal quality degradation in accuracy caused by changes in data over time, also known as concept drift [17, 20, 37]. Thus, concept impacts the reliability of failure prediction models [21]. In this chapter, concept drift refers specifically to data changes that degrade the accuracy of failure prediction models.



To overcome the effects of concept drift, previous work proposes to periodically retrain (update) failure prediction models [17, 21, 37]. However, this comes with both computational effort [18] and hidden deployment costs such as compliance verification [79] and integration with a larger software system [9]. Lyu et al.[18] propose retraining only when a label-dependent drift detector identifies a significant accuracy drop. However, this approach assumes immediate access to true labels, which is often impractical, as labels may require manual annotation by engineers once the root cause of failure is understood[38].

Continuously monitoring failure prediction models by verifying whether concept drift occurs is a promising solution [17, 37] since practitioners can understand when the model's outcome is reliable and can be used for decision making purposes [21]. However, using a label-independent drift detector could eliminate the necessity of gathering true labels. Two label-independent techniques to monitor data against concept drift in ML systems were proposed by researchers from industry [9, 39, 40], namely monitoring the skewness of the features and monitoring changes in data distribution over time. Similarly to [18], these techniques can be used to identify when the model requires updating [9, 80]. However, their effectiveness in preserving the accuracy of the failure prediction model has, to the best of our knowledge, not yet been assessed.

The contributions of this study are threefold. (1) We assess existing label-independent concept drift detection techniques on three popular open-source failure prediction datasets. (2) We propose a novel feature-reduction technique based on the model's feature importance ranking that can be incorporated with an existing unsupervised data distribution-based drift detector. (3) We present a replicable evaluation pipeline for identifying suitable unsupervised drift detectors for specific failure prediction models[1]. In this work, we answer the following research questions:

**RQ1:** Can monitoring the skewness in features individually indicate concept drift?
- (a) Can changes in specific features indicate concept drift?
- (b) Is the proportion of changing features an indicator of concept drift?

**RQ2:** Can monitoring changes in data distribution over time indicate concept drift?
- (a) To what extent can data distribution-based drift detectors identify concept drift?
- (b) What is the effect on the failure prediction models' accuracy and costs (retraining and label costs) of retraining based on a data distribution drift detector vs. periodically?

[1]Replication Package

## 3.2 Related Work

**Concept Drift Definition and Evaluation:** Concept drift generally refers to changes in the data that occur over time [14] and can be monitored using concept drift detectors [19]. Multiple drift detectors were proposed for classification problems [14]. Supervised drift detectors monitor model accuracy changes, while unsupervised drift detectors monitor data characteristics, such as the data distribution [19]. Although supervised detectors indicate degradation in the model's performance, they need immediate labels, which is sometimes impractical in real-world applications. Unsupervised detectors, which don't need labels, are, thus, more suitable.



Concept drift detectors are evaluated in two manners: *assessing the drift detection accuracy* and *assessing the performance of an ML model over time*. Drift detection accuracy measures the ability to distinguish between drifts and non-drifts, but requires knowledge of the exact drift occurrence [19, 41], which is often unknown in real-world scenarios. Previous work proposed a technique that uses a two-proportion Z-test on the error rate to label testing batches into drift and non-drift, which allows the evaluation. Assessing the performance of an ML model over time is usually preferred in the real world since knowledge regarding the moment of drift occurrence is not required [14]. This assessment technique compares the performance (e.g. accuracy, ROC-AUC, etc.) of periodically retrained ML models to those retrained based on drift detection to understand whether the drift detector can be used as a retraining indicator [19].

**AIOps Models Degradation due to Concept Drift:** AIOps models enhance software delivery and quality[20] but suffer from performance degradation over time due to concept drift [17, 18, 20, 25, 37]. For example, defect prediction models trained on past data do not generalize well to future data [17, 81]. This is a consequence of changes in the operational data (concept drift) caused by uncontrollable factors such as user workloads or hardware/-software upgrades. Monitoring and periodically updating these models is necessary to mitigate concept drift[17, 25, 37]. The effects of retraining failure prediction models based on supervised detectors have been previously researched [18] while retraining failure prediction models based on unsupervised drift detectors received less attention.

**Unsupervised Concept Drift Detection:** Industry practitioners use unsupervised data monitoring techniques to identify data changes/concept drift [9, 39, 40]. Best practices include **monitoring the skewness of features** over time [39] and **monitoring data distribution** over time [9, 39, 40]. The skewness of features can be monitored by assessing changes from training to testing data for each individual feature or the percentage of skewed features [39]. Industry practitioners also recommend monitoring the data distribution over time by comparing the estimated data distribution from the training and testing features [39, 40, 82, 83]. However, the effectiveness of these techniques has not been assessed previously in failure prediction models. Research on unsupervised drift detection [14, 19, 41] identifies two types of change detection techniques: statistical tests and distance-based drift detectors. Statistical tests identify drift by checking whether there is a significant difference between the data distribution from the train set and the test set. Distance-based detectors measure the distance between these distributions and detect drift if the distance exceeds a predefined threshold [41].

## 3.3 Data and Failure Prediction Models

We employ three publicly available AIOps datasets, the Backblaze Disk Stats Dataset, the Google Cluster Traces Dataset, and the Alibaba GPU Cluster Trace Dataset. These datasets were previously used to build failure prediction models [17, 18, 37].

The **Backblaze Disk Stats Dataset** [84] contains information about various types of hard disk drives from different manufacturers [85]. It was used to design disk failure prediction models [17, 26, 85]. Similar to previous work [17, 37] we are using around 7M data samples corresponding to 12 months of data collected during 2015.



The **Google Cluster Traces Dataset** [86] contains information about traces extracted from real-world large cluster systems. It was used [17, 87] to design job failure prediction models. The dataset contains 625K samples and was collected for 29 days (May 2011).

The third dataset, the **Alibaba GPU Cluster Trace Dataset** [88], was recently publicly released (2021) by the Alibaba Group and contains workload traces collected from a production cluster containing over 6,000 GPUs. It was used to build a job failure prediction model [18]. The data was collected for two months, July to August 2020 [89], and contains approx. 701K samples.

**Model and Features.** Random Forests is a commonly used tree-based classifier in failure prediction models [17, 18, 26, 37, 85, 87]. In our experiments, we employ the Random Forests classifier to build failure prediction models for all three analyzed datasets. Furthermore, we include Random Forests in our experiments since they are well-researched in terms of feature importance ranking extraction [90, 91], which is a crucial part of our proposed drift detection method. We build a monthly disk failure prediction model using the Backblaze dataset, a weekly model using the Alibaba dataset, and a daily model using the Google dataset.

For all three datasets, we employ the same features as previous work [17, 18, 26, 37, 85, 87]. The exact features are described in our replication package.

**Model Building Pipeline.** All the failure prediction models studied in this chapter are built by replicating previous works' approaches [17, 18, 26, 37, 85, 87]. The first step in creating the failure prediction models is data preprocessing through scaling using StandardScaler[2]. Scaling is performed since features have varying degrees of magnitude, which affects the classification training. To mimic a realistic scenario, we fit the scaler every time on the period of data corresponding to the training data and only then apply it to the testing data. We further apply undersampling with a ratio of 1:10 to reduce the severe class imbalance in the Google and Backblaze datasets. We do not apply it to the Alibaba dataset, since the classes are relatively balanced (1:3 imbalance ratio). The last steps of building the failure prediction model are training and hyperparameter tuning, for which we use Randomized Search [18]. To avoid bias we repeat all experiments using 10 different random seeds.

**Detecting Drift/Changes in Data.** To detect data changes, we select the Kolmogorov-Smirnov statistical test given its popularity in unsupervised drift detection [14, 19, 41, 92]. This statistical test verifies the similarity between two data distributions. We did not consider distance-based drift detectors since they require users to predefine drift thresholds for each dataset.

[2]Standard Scaler

## 3.4 Experimental Design

This section describes the experimental design used to answer the two research questions. RQ1 and RQ2.a. require a drift detection accuracy assessment, which can only be performed after extracting the ground truth regarding the batches that are labeled as drift and non-drift. RQ2.b. requires a model performance preservation assessment.

### 3.4.1 Extracting Drift/Non-Drift Batches

To extract the ground truth regarding when concept drift occurs, we follow the same technique as previous work [17]. In Figure 3.1, we depict the pipeline used to identify concept drift between datasets extracted from two different periods, P1 and P2. We train an ML model using the data from the first period (P1) and test it on the data from the second period (P2). The training error rate is computed by performing a 10-fold cross-validation on Data P1 and the testing error rate is obtained by testing the model on Data P2. On the two error rates, we apply a two-proportion Z-test to assess whether there is concept drift between the datasets from two different periods:

$$Z = \frac{\epsilon_{test} - \epsilon_{train}}{\sqrt{\epsilon(1-\epsilon)(\frac{1}{n_{train}} + \frac{1}{n_{test}})}} \tag{3.1}$$

where $\epsilon_{train}$ is the prediction error rate on the training set, $\epsilon_{test}$ is the prediction error rate on the testing set, $\epsilon$ is the overall prediction error rate, $n_{train}$ is the length of the training set and $n_{test}$ is the length of the testing set.

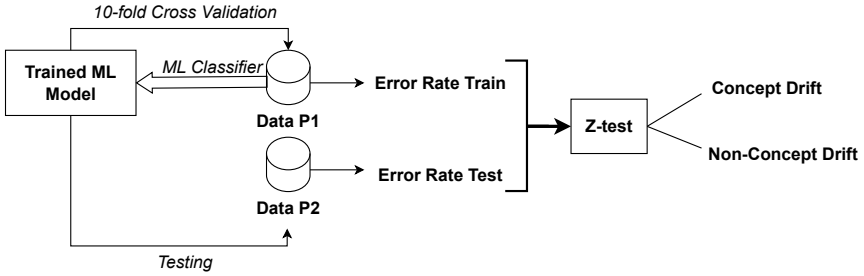



Figure 3.1: Obtaining the ground truth. Pipeline to assess the presence of concept drift between two batches (Data P1 and Data P2).

The null hypothesis of the Z-test is that there is no significant difference between the ML model's performance on datasets extracted from the two different periods, thus it is a *non-drift batch*. The null hypothesis is rejected when the p-value of the Z-test is lower than 0.05, suggesting that there is a *drift batch*.

### 3.4.2 Monitoring the Skewness of Features Individually

In this work, we monitor the skewness of features by evaluating whether either *monitoring changes in features individually* or *the percentage of features that change over time* can indicate that a batch is labeled as drift or non-drift.

**Monitoring changes in features individually.** We apply the Kolmogorov-Smirnov statistical test on two consecutive periods of the same individual feature to assess how

each feature is changing over time. We determine whether every two consecutive periods corresponds to a drift batch or a non-drift batch. We further count the number of times the feature changes in a drift batch and non-drift batch respectively. With this experiment, we aim to understand whether changes in specific features are more associated with drift or non-drift batches. If changes in one feature are more linked to drift batches than non-drift batches, monitoring that feature could indicate unsupervised model performance degradation for failure prediction models.

**Monitoring the percentage of features that change over time** With this evaluation strategy, we aim to understand whether the number of features that change from one period to another can be an indicator that the model's performance is degrading. Therefore, we apply the Kolmogorov-Smirnov statistical test on two consecutive periods of each individual feature. We count how many features have changed between these two periods and we determine whether the associated batch is labeled as drift or non-drift. This evaluation strategy assumes that, if more features are changing between two periods, there is a higher chance of having a drift.

Thus, we aim to understand whether AIOps practitioners can use the number of features that change as an unsupervised model degradation indicator. We further support our observation by analyzing the correlation coefficients between the number of features that change and the batch label (drift/non-drift), which should be close to 1 or -1 to suggest that the number of features that change is a good indicator of drift.

### 3.4.3 Monitoring Changes Data Distribution

#### Assessing the Drift Detection Accuracy of Drift Detectors

The state-of-the-art techniques identify drift by monitoring how the distribution of the data in the training set is changing compared to the distribution of the data in the testing set [41]. In terms of the number of features included to derive the data distribution, we analyze two data distribution drift detection techniques. The first technique we refer to as *KS_ALL* identifies drift by estimating the data distribution from all the features used to train the failure prediction model [41]. Concept drift detection research suggests that feature reduction through Principal Component Analysis (PCA) should be initially applied to the features before estimating the distribution for a more accurate drift detection [54], [41]. Therefore, the second evaluated technique referred to as *KS_PCA* initially reduces the dimensionality of the features using PCA and then estimates the data distribution.

We propose a *model-driven unsupervised drift detector* that takes into account the most relevant features of the model while making predictions. Our technique includes a feature reduction that computes the data distribution of solely the features that are relevant to the model. With every model training, we compute the feature importance (FI) ranking, sort the features based on their ranking in importance, and select the features whose importance value is higher than the mean importance values. Therefore, we estimate the data distribution of solely the most important features and apply the KS statistical test to identify drift. Similar to previous techniques, drift is detected when the null hypothesis of the KS statistical test is rejected. We are referring to this technique as *KS_FI*.

To extract the feature importance ranking we employ techniques previously used for classifier predictions explainability [90, 93, 94]. The most common metric to compute the feature importance (FI) ranking is the mean decrease in impurity (MDI) [93, 95, 96] also

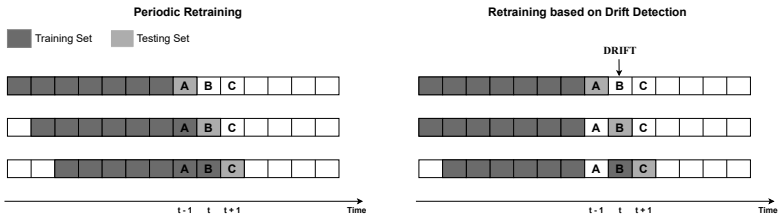



Figure 3.2: Retraining periodically vs. retraining based on drift detection strategies.



known as Gini importance. For each feature, this technique calculates the total decrease in impurity (loss) computed for each random split [93].

**Evaluation Metrics.** We employ three metrics in our evaluation strategy, the *True Positive Rate (TPR)*, the *True Negative Rate (TNR)*, and the *Balanced Accuracy*. The TPR shows the percentage of correctly identified drifts, the TNR shows how many non-drifts are correctly identified and the balanced accuracy shows the overall correctly classified drifts and non-drifts. The best value of each metric is 1.0.

### Assessing the Effects of Retraining based on Drift Detection

We aim to understand the effect of retraining based on drift detection vs. retraining periodically, which is the current state of practice for failure prediction models [17, 37]. In Figure 3.2 we illustrate the difference between the two retraining techniques. In the case of periodic retraining, every time new data becomes available the model is retrained. The new data is included in the retraining data and the old data is discarded, a retraining strategy called the sliding-window approach, also used in previous work [37]. The drift detection-based retraining strategy comes with the assumption that if no drift is detected, the new available data is similar to the one that the model is already trained on, thus retraining the model on the new data does not necessarily bring new information. Therefore, in this evaluation strategy, retraining is performed solely when drift is indicated by a drift detector. As depicted in Figure 3.2, only batch B, where the drift was identified, is included in the training set while batch A is not included since there is no drift identified. This evaluation strategy helps in reducing not only the number of times the model requires retraining, but also the costs of obtaining labels for retraining. Thus, from our example, the true labels from AIOps practitioners are not required for batch A. In terms of drift detection, we employ the same data distribution drift detection techniques as previously, namely KS_All, KS_FI, and KS_PCA. Furthermore, for this experiment, we use a model that is never retrained, which we refer to as *static model* as a baseline.

**Evaluation Metrics** We evaluate the effects of retraining based on drift detection compared to periodical retraining using three metrics. The first metric, ROC_AUC is related to the performance of the failure prediction model's performance. This metric shows how well these models distinguish between failures and non-failures. The other two metrics, the effectiveness per unit of retraining cost (ERC) and the effectiveness per unit of labeling cost (ELC) are derived from cost-effectiveness analysis [97]. The ROC_AUC takes values between [0, 1], where 1 corresponds to perfect prediction.

The ERC metric was proposed and used by previous work [18] to determine the benefits of different model retraining strategies for failure prediction models. This metric is

calculated using Equation 3.2, where the portion of retrainings refers to the percentage of time periods that require updates, while the performance improvement is the percentage of ROC AUC improvement over the static model. A higher ERC corresponds to a more cost-effective strategy.

$$ERC = \frac{Performance\_Improvement}{Portion\_of\_Retrainings} \tag{3.2}$$



Although the ERC metric can be used to determine the cost-effectiveness with respect to the retraining frequency, it does not take into account the costs of labels. Therefore, we propose another cost-effectiveness metric, namely the ELC, which is defined by Equation 3.3. Similarly to ERC, the performance improvement shows the improvement of ROC AUC percentage over the static model, while the portion of labels represents the number of required labels to perform drift detection-based retraining divided by the total number of labels that are used to perform periodic retraining. Therefore, this metric determines how much the performance improves with respect to the label annotation costs. A higher ELC corresponds to a more cost-effective strategy.

$$ELC = \frac{Performance\_Improvement}{Portion\_of\_Labels} \tag{3.3}$$

## 3.5 Experiments

### 3.5.1 Monitoring the Skewness of Features Individually

**Monitoring changes in features individually.** This experiment aims to answer *RQ1.a.* In Figure 3.3 we depict the feature change rate for each individual feature used for each failure prediction model. This rate is calculated by dividing the number of times a feature changed in a drift/non-drift batch by the total number of drift/non-drift batches. A feature change rate of 100 shows that this feature has changed in all the drift/non-drift batches, while a feature change rate of 0 shows that this feature has never changed.

The idea behind this experiment is to discover whether specific features change only in drift batches since changes in those features can indicate drift. From Figure 3.3 we can observe that in the Google and Alibaba datasets, some features only change in drift batches, but this trend does not apply to the Backblaze dataset. However, the "Smart 193 Raw Diff" feature from Backblaze is changing more in drift batches compared to non-drift batches. This shows that changes in "Smart 193 Raw Diff" can indicate drift, but it should not be used as the only drift indicator it might raise false positives when detecting drift. Furthermore, we can see that the majority of features do not change over time in the Backblaze dataset.

For the Google data, our experiments suggest that most of the features are changing in a similar proportion for both drift and non-drift batches. The only exception is the feature "Diff Machine" which has only changed within drift batches with a feature change rate of 0.06.

In the case of the Alibaba dataset, 6 out of 12 features only change in drift batches, namely "Avg GPU Work Mem", "Avg Mem", "CPU Usg", "GPU Work Util", "Max GPU Work Mem" and "Max Mem". These features refer mostly to the used resources when predicting job failures, such as GPU, CPU, or memory. However, due to the limited data availability in Alibaba, we only have one batch representing non-drift.

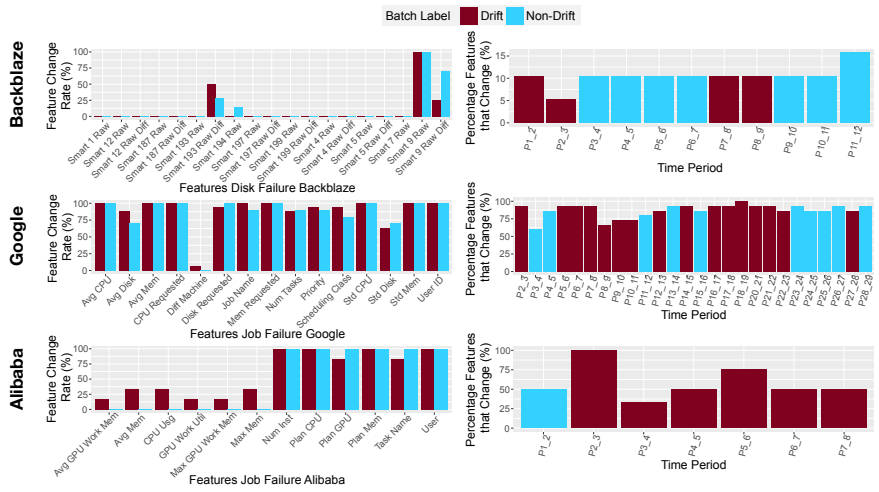



Figure 3.3: (Left column) Feature Change Rate (in percentage) in Drift/Non-Drift Batches. (Right column) Ground Truth. Batches that contain drift and non-drift for all 3 datasets.

**Monitoring the percentage of features that change over time.** This experiment aims to answer *RQ1.b*. We depict our results regarding the effect of monitoring the percentage of features that change over time in Figure 3.3.

Our findings indicate no clear distinction between drift batches and non-drift batches when evaluating the percentage of features that change from one period to another across all three datasets. Figure 3.3 shows that in some cases (e.g. drift batches M3_4 of Backblaze, P8_9 of Google, and W2_3 of Alibaba) drift batches have a lower percentage of features that change compared to non-drift batches (e.g. non-drift batches M11_12 of Backblaze, P28_29 of Google and W1_2 of Alibaba). This observation is supported by low correlation coefficients obtained when correlating the percentage of features that change with the drift/non-drift batch label.

### 3.5.2 Monitoring Data Distribution

**Assessing the Drift Detection Accuracy of Drift Detectors.** With this experiment, we answer *RQ2.a*. We assess how well each drift detection technique is able to identify the testing batches corresponding to drift or non-drift. In this manner, we assess the drift detection accuracy for each detector.

We show our results in Table 3.1, where we can notice that data distribution techniques accurately detect the non-drifts in the Backblaze disk dataset, while in the job data (Google and Alibaba) they accurately identify drifts. None of the techniques detects the only non-drift batch (W1_2) in the Alibaba dataset. The KS_PCA achieves the highest score (0.90) when detecting non-drifts in both Google and Backblaze datasets. The fact that KS_PCA outperforms KS_ALL in identifying non-drifts suggests that reducing the dimensionality of the features when detecting drift prevents an excessive false positive rate.

**Assessing the Effects of Retraining based on Drift Detection.** In this experiment, we assess the effects of including a drift detector in the maintenance pipeline of failure

Table 3.1: Drift detection accuracy metrics for the three drift detectors (KS_ALL, KS_PCA and KS_FI). With **bold** we depict the highest value for each metric for each dataset.

| | Metric | KS_All | KS_FI | KS_PCA |
|---|---|---|---|---|
| **Backblaze** | Balanced Accuracy | 0.65 | 0.01 | **0.73** |
| | True Negative Rate | **0.09** | 0.05 | 0.05 |
| | True Positive Rate | 0.80 | 0.78 | **0.90** |
| **Google** | Balanced Accuracy | **0.63** | 0.52 | 0.52 |
| | True Negative Rate | **0.94** | 0.82 | 0.65 |
| | True Positive Rate | 0.10 | 0.00 | **0.30** |
| **Alibaba** | Balanced Accuracy | **0.57** | 0.28 | 0.28 |
| | True Negative Rate | **0.66** | 0.33 | 0.33 |
| | True Positive Rate | 0.00 | 0.00 | 0.00 |



prediction models on their performances with the purpose of answering *RQ2.b*. We simulate a maintenance scenario in which the failure prediction model is retrained only when drift is indicated by one of the three evaluated drift detectors, saving both retraining times costs and retraining true labels costs. The results are summarized in Table 3.2, where we present five retraining techniques, *Static*, *Periodic* and the three drift detection retraining techniques, *KS_All*, *KS_FI* and *KS_PCA*. The static retraining technique refers to the situation in which the model is never retrained. The periodic retraining technique refers to the situation in which the model is retrained periodically. We use Static and Periodic as our lower and upper baselines in terms of ROC_AUC. The other three drift detection retraining techniques refer to the situations in which the model is retrained every time the unsupervised drift detector, KS_All, KS_FI, and KS_PCA respectively, indicate the need to retrain.

From Table 3.2 we can notice that retraining based on a drift detector with feature reduction (either PCA or our proposed technique based on feature importance) preserves the performance of the model over time. These retraining strategies achieve similar performance with periodic retraining in the cases of Backblaze and Google (96% and 83% respectively) and similar performance in the case of Alibaba (61% compared to 62% achieved by periodic retraining).

KS_FI obtains the highest ERC score for Backblaze (0.042 and 0.090) Alibaba, making it the best technique for optimal performance with minimal retraining. Furthermore, it also achieves the highest ELC score for Backblaze (0.042), showing that using this retraining strategy is the best compromise between the performance and the required number of labels. KS_PCA yields similar scores for Backblaze but with higher retraining and labeling costs.

The KS_PCA retraining technique is the most efficient from the model performance perspective for Backblaze and Alibaba, achieving high ELC scores (0.042 and 0.102, respectively). For Alibaba, it requires the fewest retraining times (25%) and labels (20%), with only a 2% ROC_AUC loss compared to Periodic. KS_ALL is the best retraining technique for the Google dataset, with the highest ERC (0.070) and ELC (0.069) scores. However, KS_ALL is too sensitive when detecting drift for Backblaze, achieving similar performance with Periodic, and ineffective for Alibaba, achieving similar performance with Static.

Table 3.2: Effectiveness retraining based on drift detection vs. static (lower baseline) and periodic retraining (upper baseline), measured in terms of the performance of the model (ROC_AUC), percentage of retrainings required (Retrainings), Effectiveness per unit of Retraining Costs (ERC), percentage of labels required (Labels), and Effectiveness per unit of Label Costs (ELC). Best results (highest ROC_AUC, ERC, and ELC and lowest Retrainings and Labels) out of the 3 drift detection techniques in **bold**.

| | Strategy | ROC_AUC | Retr.(%) | ERC | Labels(%) | ELC |
|---|---|---|---|---|---|---|
| **Backblaze** | Static | 0.92 | - | - | - | - |
| | KS_All | **0.96** | 100 | 0.040 | 100 | 0.040 |
| | KS_FI | 0.95 | **78** | **0.042** | **72** | **0.042** |
| | KS_PCA | **0.96** | 95 | **0.042** | 95 | **0.042** |
| | Periodic | 0.96 | 100 | 0.040 | 100 | 0.040 |
| **Google** | Static | 0.77 | - | - | - | - |
| | KS_All | **0.83** | **86** | **0.070** | **87** | **0.069** |
| | KS_FI | 0.81 | 89 | 0.036 | 84 | 0.047 |
| | KS_PCA | **0.83** | 95 | 0.057 | 89 | 0.062 |
| | Periodic | 0.83 | 100 | 0.060 | 100 | 0.060 |
| **Alibaba** | Static | 0.58 | - | - | - | - |
| | KS_All | 0.58 | 0 | 0.000 | 0 | 0.000 |
| | KS_FI | **0.61** | 35 | **0.090** | 35 | 0.086 |
| | KS_PCA | 0.60 | **25** | 0.080 | **20** | **0.102** |
| | Periodic | 0.62 | 100 | 0.040 | 100 | 0.040 |

## 3.6 Discussion

In this section, we answer each research question and discuss the findings resulting from our experiments.

**Monitoring the Features Individually.** Some features are linked with concept drift, but they should not be used alone as a drift indicator. However, monitoring the percentage of features that change is not a good indicator of concept drift for disk or job failure prediction models.

We demonstrate that changes in specific features (RQ1.a) can be linked with drift. Still, it cannot be used alone as a drift indicator, although considered best practice in monitoring machine learning systems [39]. Our results show that some features used to create job failure prediction models change only in drift-labeled batches. Examples are one feature, namely "Diff Machine" in Google dataset, and six features, namely "Avg GPU Work Mem", "Avg Mem", "CPU Usg", "GPU Work Util", "Max GPU Work Mem" and "Max Mem" in the Alibaba dataset. The features that change in the Alibaba dataset are related to the used resources (memory, CPU, and GPU), and for this dataset, they solely change during drift batches. However, the features related to the used resources (disk, memory, CPU) in the Google dataset change in both drift and non-drift batches. This shows that the features related to used resources are not generally an indicator of concept drift and AIOps practitioners should identify which features indicate model degradation for their AIOps models. Regarding disk failure prediction, there was no feature changing in solely drift

batches. The only feature that exhibited changes in more drift batches than non-drift batches is the "Smart 193 Raw Diff" (Load Cycle Count) feature. However, our results indicate that using this feature as a concept drift indicator leads to many false alarms.

Monitoring the proportion of features that changed was not linked with concept drift for either of the evaluated AIOps datasets. This shows that although considered best practice in concept drift monitoring [39], it cannot indicate drift in failure prediction datasets (RQ1.b). Therefore, we recommend practitioners investigate other unsupervised drift detection techniques.



**Monitoring the Data Distribution over Time.** Some data distribution-based drift detectors can accurately identify drifts, but which detector to employ is dataset-dependent. Retraining based on unsupervised data distribution drift detectors is beneficial, obtaining similar performance with periodic retraining and lowering the retraining and label costs.

In our last experiment, we simulate the scenario in which a failure prediction model is deployed into production and evaluated periodically on the upcoming batches. We compare the situation in which the model is never retrained (Static), the situation in which the model is retrained based on one of the three unsupervised drift detectors, and the situation in which the model is retrained periodically. Our results demonstrate that retraining based on unsupervised drift detectors is promising since it achieves similar performance to periodic retraining and lowers both the number of retraining times and the number of required true labels. Our findings suggest that employing unsupervised drift detectors as data monitoring tools is a promising strategy to lower the retraining labels' costs while preserving accuracy (RQ2.b).

Another important conclusion drawn from our results is that no unsupervised drift detector achieves the best results on all three analyzed datasets (RQ2.a). This shows that choosing the most suitable drift detector depends on the AIOps application and dataset. Therefore, AIOps practitioners have to experiment with their datasets to identify the most suitable unsupervised drift detection technique.

In our experiments, we evaluate both the accuracy of drift detection (RQ2.a.) and the effects of retraining based on drift detection (RQ2.b) to understand whether we can link drift detection accuracy to its effects when used as a model retraining indicator. This approach allows practitioners to identify the suitable drift detector for their datasets by evaluating drift detection accuracy on their training data. However, our results suggest that the behavior of the unsupervised drift detection techniques is different in the two evaluation scenarios. For instance, the KS_All drift detector identified 66% of the drifts on the Alibaba dataset in the former evaluation scenario (Table 3.1), while in the latter scenario (Table 3.2) it was not able to detect any drift. Thus, the accuracy of drift detection on smaller batches does not reflect how the detector is behaving in a production environment. Therefore, we suggest that practitioners allocate a testing period for selecting the most appropriate drift detector. In this testing period, practitioners can employ our proposed pipeline and carefully analyze the impact and required costs of retraining based on each drift detector while compared to never retraining the model (lower baseline) and periodically retraining a model (upper baseline).

## 3.7 Threats to Validity

An **external threat to validity** is generalizability, as we used only three publicly available datasets (Backblaze, Google, Alibaba) and focused on solely existing failure prediction models without exploring techniques to improve the model (e.g. adding a new feature). However, the dataset samples are representative of real-world machines since they are published by well-known organizations for research purposes. For **internal validity**, we split data into periods and train-test sets consistently with previous work [17, 18, 37]. Regarding **construct validity**, we used Randomized Search for hyperparameter tuning, fixed iteration time to 100, applied undersampling to achieve a 10:1 ratio of non-failure to failure samples, and used 10 random seeds to minimize bias. For feature selection in KS_FI, we chose features with importance above the mean.



## 3.8 Conclusions and Future Work

The main goal of this article is to understand to what extent unsupervised data monitoring tools can be employed in real-world failure prediction models to identify concept drift. Furthermore, we aim to quantify the benefits of employing an unsupervised drift detector in the maintenance pipeline of a failure prediction model in terms of the number of retrainings and label costs with respect to the model performance trade-offs. To do so we extracted the best practices in unsupervised techniques for monitoring machine learning systems suggested by industry practitioners, such as *monitoring the skewness of features over time* [39] (*monitoring the percentage of features that change* and *monitoring the skewness of features over time*) and *monitoring the data distribution over time* [9, 39, 40]. We applied them to three failure prediction models and verified how well they indicated concept drift (model degradation). The employed datasets are representative of real-world data since they were either provided by real-world organizations (Google and Alibaba) or contain data collected from various hardware devices (Backblaze) [26].

We empirically show that monitoring the percentage of the features that change is not correlated with the presence of drift. Our experiments suggest that some features can be linked to the presence of drift, but they cannot be used alone as drift indicators. However, out of the best practices in unsupervised monitoring machine learning systems, monitoring the data distribution is the most promising technique. Therefore, unlike previous work proposing periodic model retraining [17, 37] or retraining based on supervised drift detectors [18], we demonstrate that unsupervised data distribution-based drift detectors effectively indicate when to retrain failure prediction models, reducing retraining and labeling costs. Furthermore, in this chapter, we proposed integrating a feature importance technique extracted from the model into data distribution-based drift detectors. This technique extends beyond Random Forests and can be applied to any classifier where key features can be extracted, making it widely applicable. The feature importance based detector (KS_FI) was the most cost-effective retraining for Backblaze and Alibaba datasets. Furthermore, we address the model monitoring research gap [98] by proposing a pipeline that helps AIOps practitioners evaluate unsupervised drift detectors and select the most suitable one to monitor an AIOps model. Our experiments establish a foundation for evaluating drift detectors in the MLOps monitoring pipeline of failure prediction models.

**Future Work:** Given the promising results of unsupervised drift detectors, as future

work, we aim to expand our drift selection pipeline into a framework that AIOps practitioners can employ to identify the most suitable unsupervised drift detectors for their datasets or applications. However, we do not recommend a specific drift detector since some applications prioritize capturing more drifts, while others aim to reduce false alarms. We seek to expand our analysis to other AIOps applications such as node failure prediction or incident prediction.

# 4

# Sustainable Machine Learning Retraining: Optimizing Energy Efficiency Without Compromising Accuracy



*The reliability of machine learning (ML) software systems is heavily influenced by changes in data over time. For that reason, ML systems require regular maintenance, typically based on model retraining. However, retraining requires significant computational demand, which makes it energy-intensive and raises concerns about its environmental impact. To understand which retraining techniques should be considered when designing sustainable ML applications, in this work, we study the energy consumption of common retraining techniques. Since the accuracy of ML systems is also essential, we compare retraining techniques in terms of both energy efficiency and accuracy. We showcase that retraining with only the most recent data, compared to all available data, reduces energy consumption by up to 25%, being a sustainable alternative to the status quo. Furthermore, our findings show that retraining a model only when there is evidence that updates are necessary, rather than on a fixed schedule, can reduce energy consumption by up to 40%, provided a reliable data change detector is in place. Our findings pave the way for better recommendations for ML practitioners, guiding them toward more energy-efficient retraining techniques when designing sustainable ML software systems.*

This chapter is based on the following peer-reviewed publication:

*Lorena Poenaru-Olaru, June Sallou, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. 2025. Sustainable Machine Learning Retraining: Optimizing Energy Efficiency Without Compromising Accuracy. Proceedings of the 11th International Conference on ICT for Sustainability (ICT4S), Dublin, Ireland, 2025* [45].

## 4.1 Introduction

The increasing adoption of Machine Learning (ML) and Artificial Intelligence (AI) within organizations has resulted in the development of more ML/AI software systems [29]. Although ML/AI brings plenty of business value, it is known that the accuracy of ML applications decreases over time [39]. Thus, ML developers must monitor and maintain their ML systems in production. One reason for this phenomenon is the fact that ML applications are highly dependent on the data on which they have been trained. Real-world data usually changes over time [14] – a phenomenon often referred to as concept drift [15] – which can significantly impact the normal operation of ML systems [40]. Therefore, appropriate maintenance techniques are required for the design of ML software systems. One common approach to maintaining these systems is to periodically update these applications by retraining the underlying ML models with the latest version of the data [17], [37].



On another note, the process of training machine learning models has raised substantial concerns about the carbon footprint of ML applications [99, 100]. For this reason, regularly retraining ML applications to preserve their accuracy, implies considerable energy consumption [44]. On the other hand, not retraining ML applications at all can severely impact their performance over time [14], affecting their reliability in practice. Therefore, there is a need for sustainable ML retraining methods that reduce the environmental impact of ML applications [29, 44].

Previous research [101] has explored the energy consumed during training and inference of continuous learning, which are ML applications where a machine learning (ML) model learns incrementally without requiring full retraining. Continual learning ML applications present significant challenges when incorporating domain experts' feedback into the ML model [15]. Consequently, a continual learning approach is not suitable for certain applications. A better solution is periodically retraining ML models after accumulating sufficient new samples. However, to the best of our knowledge, the effects of retraining an ML model using various retraining techniques on both accuracy and energy consumption have not been thoroughly examined. Therefore, this study aims to examine the effects of various retraining techniques on accuracy and energy consumption in real-world ML applications, to provide best practices for designing sustainable ML systems. We analyze these retraining techniques through the lens of two perspectives, namely the *data perspective* and the *frequency perspective* [44]. The *data perspective* addresses the impact of the data included in the retraining process, while the *frequency perspective* focuses on the impact of the frequency at which a model is retrained. Moreover, since some ML applications also require substantial energy for inference [30], we also explore whether the retraining strategy affects the energy consumption of the inference tasks.

In this empirical study, we use failure prediction applications as the case study ML application. Failure prediction applications are part of the AIOps research domain and are ML applications that aim to identify failures in large and complex software systems in order to improve their efficiency and reliability [20]. We specifically select failure prediction applications since periodic retraining is commonly applied to keep models up to date and these applications require large datasets, making them considerably energy intensive in practical settings [8, 17, 18, 36, 37]. We argue that an efficient model monitoring and retraining strategy can lead to significant improvements in maintaining these models.

Hence, we employ three failure prediction models previously presented in the literature, which are built using three open-source real-world datasets.

Previous research has compared retraining failure prediction models using all available data versus only the most recent data, with both approaches maintaining performance over time [37]. However, retraining on all data is likely to consume more energy [31, 44]. Our study empirically evaluates the energy consumption and compares the accuracy and energy efficiency of these two retraining techniques.

Related work also suggests that periodic retraining is less sustainable compared to drift-based approaches [29, 44]. However, drift detectors consume energy [102] and can be overly sensitive [41], triggering unnecessary retraining and increasing energy consumption. Hence, in this work, we quantify the impact of drift detection-based retraining on the energy efficiency of AI-driven failure prediction systems.

The main contribution of this study is a quantitative empirical study of the impact of different retraining techniques on the trade-off between accuracy and energy consumed by ML systems during both training and inference. Our research shows that drift-based retraining approaches reduce the energy consumption of failure prediction models over time. However, the gains in energy efficiency are highly dependent on the choice of a drift detector. All experiments and analyses are openly available in a replication package[1].



## 4.2 Background

In this section, we introduce the background knowledge on which the remainder of the paper builds, i.e., AIOps and failure prediction, concept drift, and retraining approaches.

### 4.2.1 AIOps

Large-scale software systems generate vast amounts of operational data, making manual analysis and inspection impractical [20]. The term AIOps, introduced by Gartner [103], refers to applying ML techniques to automate this process [17]. AIOps applications are ML applications used to monitor large systems and enhance software delivery, compliance, quality, and security in organizations [20]. A recent survey identifies four key AIOps categories: root-cause analysis, incident detection, failure prediction, and automated actions [104]. This work focuses on failure prediction, specifically analyzing two ML applications: disk failure prediction and job failure prediction, which learn patterns from past failures to anticipate hardware failure (disk) or software failure (job). It has been previously shown [17, 18, 26, 37, 85] that these applications are severely affected by concept drift and that they employ large amounts of data to train.

### 4.2.2 Concept Drift

The term concept drift refers to changes in the data over time [19]. It is a ubiquitous phenomenon in real-world data, as data changes are generated by uncontrollable external factors [41]. Concept drift can severely impact the accuracy/performance of ML models over time since the ML algorithms used to build these models work under the assumption that the distribution of the data data learned during the training process should be similar to the distribution of the data on which the model is evaluated [14]. However, in the real

[1]Replication Package

world, this assumption often does not hold, since data are continuously changing, which can lead to noticeable drops in the model's accuracy over time.

Continuous model update/retrain is a commonly known technique to mitigate the effects of concept drift on ML models over time [15, 58, 105].

From the **retraining frequency perspective** researchers [15] have distinguished between two retraining techniques, namely *periodic retraining* and *informed retraining*. Periodic retraining implies that models are retrained based on a predefined period, while informed retraining means that a model is retrained based on a data monitoring tool called a concept drift detector. In the latter situation, the concept drift detector evaluates whether the training data becomes significantly different than the data the model is evaluated on [41].

From the **retraining data perspective**, there are two techniques previously presented in the literature, namely the *sliding window* and the *full-history* retraining approach [18, 37, 47]. The sliding window approach implies that the model is retrained only on the newest data, discarding old samples, while with a full-history retraining approach, the model is retrained on all the available data until a certain point in time. Therefore, the latter retraining technique constantly enriches the training dataset with new samples once they become available.

## 4.3 Related Work

### 4.3.1 Concept Drift in AIOps Applications

Previous work has shown that multiple ML applications within AIOps, including failure prediction model, have been affected by concept drift [8, 17, 20, 36, 47, 106]. In failure prediction applications, concept drift can be caused by different external factors, such as feature updates, user workloads, or software/hardware updates [8]. These factors can affect the behavior of the data over time, which usually impacts the accuracy of failure prediction models. This aspect makes concept drift a serious threat to the trustworthiness of failure prediction models, especially when their output is used in decision-making processes [20].

When it comes to AIOps applications, previous works recommend AIOps practitioners to periodically retrain failure prediction models [8, 17, 37]. From the retraining data perspective, both the sliding window [8, 17, 18, 25, 37, 47, 107] and full-history approaches [18, 37, 47] retraining approaches were employed to update AIOps models over time. From the retraining frequency perspective, the most commonly used retraining technique is the periodic retraining [8, 17, 18, 25, 37]. However, Lyu et al. [18] also experimented with informed retraining. The authors employed a supervised concept drift detector to monitor the error of the failure prediction model over time and retrained every time they observed a significant error increase. To compute the error, the ground truth, also known as true labels, is required. True labels refer to which sample is a "failure" and which sample is a "non-failure". However, in some AIOps applications, continuously monitoring the error over time might not be possible since obtaining true labels is expensive in time and resources. For these applications, to obtain the true labels operational engineers have to continuously perform root cause analysis to understand the main cause of the failure[20, 47, 106]. This consumes a significant amount of the time of operational engineers which makes acquiring the true labels significantly expensive. In this situation,

employing an unsupervised drift detector that does not require computing the error using true labels would be preferred [41, 47]. Therefore, unlike previous work [18], in this chapter, we analyze the effects of retraining based on an unsupervised drift detector. Furthermore, while previous work only took into account the accuracy when determining the effects of retraining based on drift detection, we also consider the consumed energy to ensure the sustainability of the failure prediction models over time.

### 4.3.2 Sustainability of ML Systems

The adoption of ML applications in the industry has seen considerable growth over the past years [99, 100]. Although numerous domains benefit from the use of machine learning, these models consume significant amounts of energy, raising concerns among researchers about the environmental impact of such applications [29, 30, 99, 100]. This led to the rise of the GreenAI research field, which encourages the development of ML applications that consume less energy while preserving their accuracy [28].



One key practice in building GreenAI systems is reporting the energy consumption of ML applications, which could raise awareness of their carbon footprint and help find more sustainable configurations [28]. Wu et al. [100] examined the carbon footprint of Facebook's ML applications and discovered that both training and inference have a significant contribution to the overall carbon footprint of the ML application. Plenty of research has been focused on understanding the environmental impact of training ML systems, such as the study of Xu et al. [108] targeting multiple computer vision applications. On the other hand, the work of Luccioni et al. [30] analyzed inference energy in tasks like image classification and language modeling. They concluded that although tasks involving images are more energy-intensive, model training remains significantly more carbon-intensive compared to inference.

While significant attention has been given to the energy consumption of training and inference in ML models, the energy used during the model's active production phase (model lifecycle) has received less focus. Trinci et al. [101] investigated the training and inference energy efficiency of continual learning algorithms for computer vision applications, but these algorithms are not directly applicable in the AIOps context due to the high costs of continuously gathering true labels[20] to perform continual learning. AIOps applications require ongoing model monitoring and updates, but these should occur in batches at predefined intervals rather than continuously. Although previous work [44] explored sustainable retraining techniques, there is a lack of empirical analysis demonstrating their effectiveness in maintaining model accuracy and reducing energy consumption over time. Omar et al. [102] have studied the energy consumption of multiple drift detectors with respect to their drift detection accuracy. However, the drift detectors analyzed operate in a supervised manner, relying on error rate computation after inference, which requires true labels. This approach is impractical for some AIOps applications, where acquiring true labels depends on human annotation[20]. Therefore, unsupervised drift detectors are more suitable, as they identify drift by analyzing differences between training and inference data without needing true labels. Our study differs from previous work [102] by employing unsupervised drift detectors and focusing on their ability to indicate the need to retrain a failure prediction model, rather than assessing the drift detection accuracy.

## 4.4 Research Questions

The goal of our study is to assess the impact of different retraining techniques on the energy consumption of failure prediction models. With this aim, we address the following research questions (RQ):

RQ1. What is the impact of each retraining technique on the training energy consumption?
  (a) What is the impact on energy consumption of employing the sliding window vs. the full-history approach (retraining data perspective)?
  (b) What is the impact on energy consumption of employing the periodic vs. the informed retraining (retraining frequency perspective)?
  (c) What is the best retraining technique overall?

RQ2. What is the impact of each retraining technique on the inference energy consumption?



## 4.5 Methodology and Experiments

In this section, we outline the methodology used to address the research questions and conduct our experiments. Specifically, we discuss the used datasets, the process for building the failure prediction models, the experimental design, and the energy consumption measurement.

### 4.5.1 Datasets

We employ the only three publicly available open-source AIOps datasets to build failure prediction models according to previous work [17, 18]: the Backblaze Disk Stats Dataset [84], the Google Cluster Traces Dataset [86], and the Alibaba GPU Cluster Trace Dataset [88].

The **Backblaze Disk Stats Dataset** has been previously used to build disk failure prediction models [17, 18, 26, 37, 85]. It contains information about operational hard drives available in the data center collected daily since 2013. The dataset includes both drive information (manufacturer, serial number, or capacity), as well as information related to early error detection extracted through a monitoring system implemented by the manufacturer, called SMART attributes (Self-Monitoring, Analysis, and Reporting Technology). Given that data collected before 2015 does not include SMART attributes and to align with previous work [17], we employ 12 months of data collected in 2015.

The **Alibaba GPU Cluster Trace Dataset** is relatively new, released in 2021 by the Alibaba Group, and, to the best of our knowledge, there has been only one work that employed it to build job failure prediction models [18]. It is composed of information regarding job execution extracted from a large-scale data center. The data is collected for a period of 2 months, from July to August 2020 [89], from approximately 6500 GPUs across around 1800 machines.

The **Google Cluster Traces Dataset** has been previously used to build job failure prediction models [17, 18, 37, 87]. The dataset includes information about jobs executed on a large-scale cluster at Google collected for 29 days in May 2011.

### 4.5.2 Machine Learning Models

When building the failure prediction models, i.e., disk failure prediction for the Backblaze dataset and job failure prediction for the Alibaba and Google datasets, we replicate the

pipelines presented in previous work [17, 18, 26, 37, 85, 87]. Thus, we use the same features and follow the same model design strategy presented in previous studies. More details about features are specified in our replication package.

#### Labels and Features

The Backblaze dataset contains an attribute indicating whether a drive failed the day after data collection, which serves as the label for our failure prediction model. It consists of approximately 7 million samples, with 0.05% labeled as failures and 99.95% as non-failures. For the prediction model, we selected 19 temporal SMART features from previous research [17, 18, 26, 37, 85]. Of these, 11 are non-cumulative (raw) values from the last day, while 8 are cumulative (raw diff) changes over one week compared to the previous day's values.

To create the final Alibaba dataset, we remove unfinished jobs and those ending in less than five minutes, except those labeled with the status "fail." The final dataset consists of 701,000 samples, with 34.5% labeled as failures and 65.5% as non-failures. To train the job failure prediction model, we use 12 features, comprising 6 configuration (conf) features and 6 temporal (temp) features calculated over a 5-minute window since job submission.

The Google dataset lacks a direct attribute for failed or non-failed jobs. Each job (sample) can have multiple events (fail, finish, kill, submit, update, evict, schedule) and states (pending, dead, running, unsubmitted). Following previous research [17, 18, 37], a job is labeled as "fail" only if its final state is "fail." We remove jobs with incomplete records and those that finished within five minutes of submission, as they do not provide sufficient metrics for predicting failure. The final dataset includes approximately 625,000 samples, with 1.5% labeled as failures and 98.5% as non-failures. For the job failure prediction model, we use 15 features: 9 configuration (conf) features and 6 temporal (temp) features calculated over a 5-minute period since job submission.

#### Model Building Pipeline

We built three model failure prediction models, one disk failure prediction, and two job failure prediction models, corresponding to the three datasets employed in this study, Backblaze, Google, and Alibaba, respectively. To closely resemble the failure prediction models presented in previous work [17, 18, 26, 37, 85, 87] we build a monthly failure prediction model for the Backblaze dataset, a daily failure prediction model for the Google dataset and a weekly failure prediction model for the Alibaba dataset. We split each dataset in half. The first part is used to train each model, while the second part is further divided into smaller subsets, each corresponding to a specific period (day, week, or month, depending on the dataset). These subsets mimic the real-world scenario in which new data is generated periodically and are used for inference. Fig. 4.1 shows a detailed overview of our model-building pipeline.

The model-building pipeline is similar for all our three failure prediction models unless specified otherwise. The first step of the pipeline is preprocessing the data through scaling. Similar to previous work [17, 18, 37], we employ a standard scaler which we fit on the training data and apply on the inference data such that all values of the features range in a predefined interval.

The Backblaze and Google datasets suffer from high-class imbalance, meaning that the number of failure samples is tremendously lower compared to the number of non-failure samples. For the Alibaba dataset, the class imbalance is not as severe. Therefore, similar

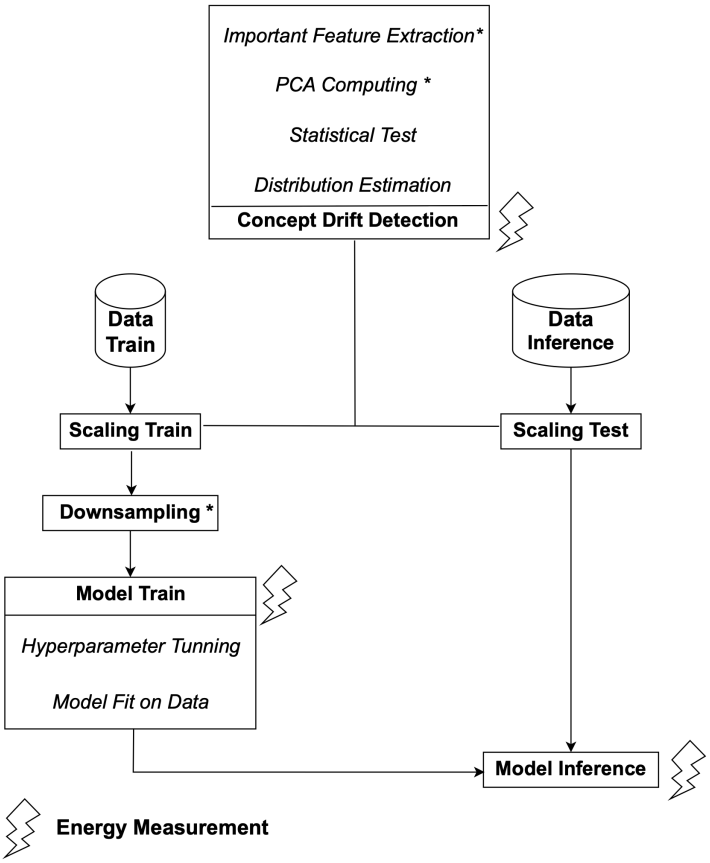



Figure 4.1: Energy measurements in model pipeline. The symbol '*' indicates that the specific step was done only when applicable.

to previous work [17, 18, 37], we perform downsampling for the Backblaze and Google datasets obtaining a class balance ratio of 1:10. This indicates that for each failure sample, there are 10 non-failure-samples.

The last steps of the model-building pipeline are model training and inference. For this step, we employ a Random Forest classifier as it is one of the most popular classifiers used in failure prediction achieving the highest performance in distinguishing between failure and non-failure samples [17, 18, 26, 37, 85]. While training the model, hyperparameter tuning is performed through Randomized Search, similar to previous work [18]. The trained model is further used for inference.

### Failure Prediction Evaluation Metric

Similar to previous work [17, 18, 37], we employ ROC AUC to evaluate the accuracy of the failure prediction model. In this chapter, we will use the terms ROC AUC and accuracy

interchangeably. ROC AUC is a metric that measures the model's ability to assign higher predicted probabilities to the positive class compared to the negative class. The highest ROC AUC score is 1.0 and shows that all samples are correctly classified. We choose this metric due to the high-class imbalance in all datasets.

### 4.5.3 Retraining Approaches

In our experiments, we investigate the energy consumption of different retraining techniques. From the retraining data perspective, we analyze the effects of two retraining techniques, namely the *full-history approach* and the *sliding window approach* [37]. In full-history retraining, the model is retrained using all data accumulated up to the given retraining time. In sliding window retraining, at each retraining step, new data is incorporated while the oldest data, corresponding to one period, is discarded. Unlike full-history retraining, this method maintains a relatively constant training dataset size.



From the retraining frequency perspective, we analyze the effects of two retraining techniques, namely the *informed* vs. *periodic* retraining [15]. In the periodic retraining approach, the model is retrained at predefined intervals, regardless of whether the data distribution has changed. To remain consistent with prior work [17, 18, 37], we perform retraining at different intervals depending on the dataset: monthly for Backblaze, weekly for Alibaba, and daily for Google. In contrast, the informed retraining approach triggers model updates only when a significant shift in the data distribution is detected. To identify such shifts, we employ the Kolmogorov-Smirnov (KS) statistical test [14, 19, 41, 92], a widely used unsupervised drift detection technique.

To further analyze the impact of different drift detection strategies on energy consumption, we explore multiple variations of drift detection. This process consists of two primary steps: first, estimating the distributions of both the training and inference data, and second, applying the KS test to compare these distributions. We examine three distinct approaches to drift detection. The first approach, KS-ALL, estimates distributions using all available model features. The second, KS-PCA, reduces the feature dimensionality such that 95% of the variance in the data is preserved using Principal Component Analysis (PCA) before estimating distributions. The third approach, KS-FI, selects only the most relevant features, filtering out those with a feature importance ranking below the mean, as determined by Gini importance. Both KS-PCA and KS-FI introduce a feature selection step before distribution estimation. While KS-PCA applies PCA for dimensionality reduction, KS-FI retains only the most informative features based on their importance ranking.

### 4.5.4 Experimental Design & Setting

In our experimentation, we study 8 retraining configurations based on two retraining perspectives: data and frequency.

We repeat our experiments to mitigate bias in our results caused by the randomness of the ML model. We allocate a one-week budget for each dataset, running each configuration multiple times. For the Backblaze and Alibaba datasets, we repeat the experiments 30 times using 30 different random seeds. The Google dataset is one of the most computationally intensive, therefore only 5 random seeds are computed. However, no significant variation in the results was observed by experimenting with different random seeds. Furthermore, when collecting energy consumption data, we shuffle the configurations to reduce the

risk of background activities or factors such as temperature impacting only a category of experiments, following recommendations for energy studies [109]. All our conclusions are verified using the Wilcoxon statistical test with a confidence level of 95%.

The experiments are run on a machine with an AMD Ryzen 9 7900X processor (12 physical cores, 24 threads), 64 GB RAM (2x32GB DDR5 @ 5.600MT/s), and an MSI Geforce RTX 4090 (24GB GDDR6X memory) graphic card. The operating system is Ubuntu 22.04.3, with Linux kernel version 6.2.0.

### 4.5.5 Energy Consumption

We measure energy at the level of the *model pipeline* and the *model lifecycle*. The model pipeline level refers to the components of the failure prediction model that are measured. The model lifecycle refers to how the total energy consumed by a certain process is consumed throughout the lifecycle of the machine learning model. In this subsection, we further describe which tool we employ to measure energy consumption.



#### Energy Measurement Tool

Energy measurement tools help quantify the sustainability of different experimental settings. To measure the consumed energy, we employ the CodeCarbon [110] Python package. The energy consumed by the CPU and RAM is measured through RAPL and the energy consumed by the GPU is measured through NVIDIA Management Library [108]. CodeCarbon measures the duration of each experiment (in seconds) and the consumed energy (in kWh). Furthermore, this energy measurement tool has been previously used to measure the energy consumed by drift detectors and the energy consumed during training and inference [101, 102].

#### Measuring Energy Consumption of the Model

In our experiments, we measure the energy consumption of multiple steps in the failure prediction model lifecycle as shown in Figure 4.1. In this work, we are solely interested in understanding the impact of different model retraining techniques on the energy consumed during the model lifecycle. Thus, we are not measuring the energy of all the steps required in building the models, such as data scaling or downsampling to balance the two classes. In our experiments, we measure three types of energy, namely *training energy*, *drift detection energy* and *inference energy*. Furthermore, in our results, we depict the cumulative values of the energy of these models. For example, the training energy is composed of the initial training of the model and energy consumed to perform retraining either periodically (if the configuration is Periodic) or when the drift detection indicates (if the configuration is drift detection based). For the Static configuration, we perform training only once and never retrain further.

The **training energy** is the total energy consumed by training/retraining. This energy measurement is composed of the energy measured during the hyperparameter tuning phase, in which the best hyperparameters are chosen to fit the training data, and the energy measured during the phase in which the best model is fitted on the training data as described in Fig. 4.1.

**Drift detection energy** refers to the total energy consumed in detecting data drift. Depending on the detector—KS-ALL, KS-PCA, or KS-FI—this energy is measured in two

or three components. KS-ALL includes energy used for estimating the data distribution and applying a statistical test to determine significant changes. KS-PCA adds a third measurement for the energy spent on dimensionality reduction using PCA. Similarly, KS-FI requires three measurements, but instead of PCA, it measures the energy used to extract and filter important features.

The **inference energy** represents the energy consumed by applying the trained model to perform inference and extract the predictions.

## 4.6 Results

In this section, we present our results and answer the research questions.



### 4.6.1 Impact on Training Energy

To answer RQ1, we depict our results in Fig. 4.2, presenting at the top the results of ROC AUC and at the bottom the total training energy combined with the total drift detection energy for each retraining technique for each dataset, when applicable. In the case of the Static model, the total training energy contains only the initial model training. When it comes to the Periodic model, there is no total drift detection energy, thus we solely depict the energy consumed during the periodical model retraining process. The drift detection energy is solely considered for the KS-ALL, KS-FI, and KS-PCA models.

#### RQ1a: Full-History vs. Sliding Window Retraining

In this experiment, we evaluate the sliding window and full-history approaches in terms of model accuracy (ROC AUC) and energy consumption (joules). To enhance the generalizability of our conclusions, we assess these retraining strategies in two scenarios: periodic retraining (Periodic) and drift-based retraining using the KS-ALL, KS-FI, and KS-PCA methods. Additionally, we include a baseline scenario with a static model (Static), trained only once, to assess the overall benefits of retraining.

In Fig. 4.2 we show that in almost all situations the full-history approach is more energy-intensive when compared to the sliding window approach. The only exception can be observed in the situation of KS-ALL for the Alibaba dataset, where there is no significant difference (p-value of 0.67) in terms of the amount of energy consumed using a sliding window or a full-history retraining approach. This exception can be explained by the fact that the ROC AUC for KS-ALL_FH is similar to the ROC AUC for KS-ALL_SW and to the ROC AUC of the Static model. This shows that in this particular situation for both full-history and sliding window approaches the drift detector did not identify any drift, and, therefore, no retraining has been performed besides the initial model training. In all other situations, the energy consumed by retraining using a sliding window approach was statistically lower than the energy consumed by retraining using a full-history approach (p-values below 0.05). For instance, when it comes to the Periodic model, we can notice an almost 25% decrease in energy consumed while retraining using the sliding window approach vs. the full-history approach.

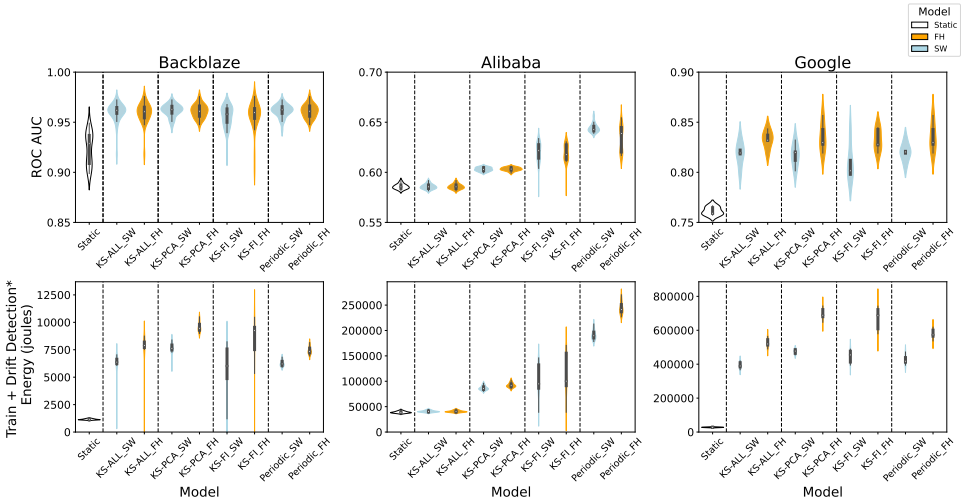



Figure 4.2: ROC AUC and energy consumption of each retraining technique. Note: for the informed retraining (KS-ALL, KS-PCA, and KS-FI all configurations) the energy consumed is calculated during both the retraining and the drift detection.

**RQ1a Answer**

A sliding window approach consumes significantly less energy than a full-history approach.

### RQ1b: Periodic vs. Informed Retraining

From Fig. 4.2 we can notice that in most cases the train energy consumption is reduced when using an informed retraining (retraining based on drift detection) technique for all datasets.

When it comes to the Alibaba dataset, the energy consumed by training based on the KS-ALL drift detector is significantly similar to the energy consumed in the Static configuration. This shows that the KS-ALL drift detector does not identify any change in the data, and thus, any need for retraining, consuming almost the same amount of energy as the Static model. However, there is a significant difference in the ROC AUC between KS-ALL and the other configurations (at least 4%), showing that retraining is required. Thus, this drift detector is not able to properly identify the drift in this dataset, and should not be used as an indicator of when to retrain. We can also notice that some informed retraining techniques consume more energy than periodic retraining techniques. For instance, retraining a model for the Google dataset using a KS-FI or a KS-PCA drift detector is more energy-intensive than periodically retraining the model. The reason for this is the fact that these drift detectors are too sensitive to changes in data and signal the need to retrain often. Since these detectors also consume energy and the number of times the model required retraining is not significantly reduced, these configurations consume more energy compared to periodic retraining. On the other hand, for the Google dataset, we can observe that some drift detectors, KS-FI and KS-PCA, are considerably sensitive

and constantly indicate the need for retraining. For this reason, they consume more energy compared to periodic retraining and this finding shows that informed retraining is not always more sustainable than periodic retraining.

Table 4.1: Percentage of energy consumed by each drift detector from the energy consumed for training + drift detection.

| Drift Detector | Retraining | Backblaze | Alibaba | Google |
|---|---|---|---|---|
| KS-ALL | SW | 3.17 | 3.90 | 0.41 |
| | FH | 2.65 | 3.91 | 0.37 |
| KS-PCA | SW | 1.77 | 1.54 | 0.26 |
| | FH | 1.50 | 1.53 | 0.21 |
| KS-FI | SW | 1.63 | 0.47 | 0.13 |
| | FH | 1.16 | 0.47 | 0.11 |



In Table 4.1, we present the percentage of the energy consumed solely by the drift detector out of the energy consumed by both training and drift detection. This table shows that the energy consumed by the drift detector is relatively low (less than 4%). This shows that incorporating an unsupervised drift detector does not bring too much overhead to the total energy consumption. We can further see from Table 4.1 that the KS-ALL drift detector consumes more energy than KS-PCA and KS-FI for all datasets. This can be explained by the fact that this drift detection technique derives the data distribution from the entire feature space, while the KS-PCA and KS-FI perform feature reduction before deriving the data distribution. Thus, unsupervised drift detectors incorporating feature reduction are generally sustainable.

**RQ1b Answer**

Informed retraining usually consumes less energy than periodic retraining when the drift detector is properly chosen.

#### Best Overall Retraining Approach

When choosing the best overall retraining approach, we have to consider which retraining technique achieves the highest ROC AUC while lowering energy consumption compared to periodic retraining. Furthermore, we consider the Static model as a lower bound in terms of ROC AUC.

To answer this research question (RQ1c), we examine the energy consumed for each retraining technique with respect to its corresponding ROC AUC improvement depicted in Figure 4.2. When it comes to the Backblaze datasets, there is no statistical difference in ROC AUC among configurations that include a drift detector (KS-ALL, KS-PCA, and KS-FI) and the periodic configuration. However, the KS-FI using a sliding window approach consumes the least energy. The KS-FI_SW is the best overall for the Alibaba dataset, achieving the highest ROC AUC with the lowest energy use. Regarding the Google dataset, the retraining technique that consumes the least amount of energy while achieving the highest performance is the KS-ALL drift detector. These experiments demonstrate that no

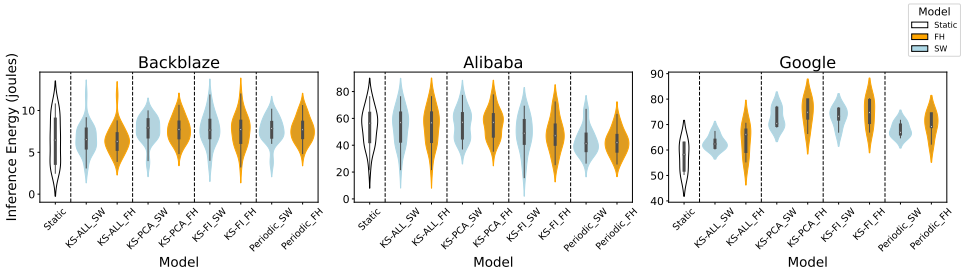



Figure 4.3: Energy consumed during inference for each retraining technique.

single retraining technique is universally optimal. Therefore, further experimentation is necessary to identify the most sustainable approach for each case.

Table 4.2: Energy estimation for using a drift detection-based retraining approach vs. a periodic approach over the period of one . In the Drift Detection column, we depict the most sustainable drift detection-based retraining technique for each dataset, namely KS-FI for Backblaze and Alibaba and KS-ALL for Google. Note: Only the sliding window retraining technique is presented since it is the most sustainable. In this table, K implies that the value needs to be multiplied by 1.000 and M implies that the value needs to be multiplied by 1.000.000

| Drift Detector | Retraining | Periodic | Drift Detection |
|---|---|---|---|
| Backblaze | 6 months | 6.3K | 5.9K |
| | 1 year estimate | 12.6K | 11.8K |
| Alibaba | 1 month | 191.3K | 106.9K |
| | 1 year estimate | 2.3M | 1.3M |
| Google | 2 weeks | 427.0K | 393.7K |
| | 1 year estimate | 11.1M | 10.2M |

Each of our datasets was evaluated during periods of different lengths, namely 6 months for Backblaze, 1 month for Alibaba, and 2 weeks for Google. These lengths are strongly dependent on the size of each dataset and on how failure prediction models were designed in previous work [17, 18, 37]. Therefore the retraining frequency for each failure prediction model is different, monthly retraining for Backblaze, weekly retraining for Alibaba and daily retraining for Google. To better understand the impact of retraining periodically vs. retraining based on drift detection over a longer period, we estimated the energy consumed on training + drift detection over one year for each dataset. In this experiment, we assume no change in the energy consumed during the given period and the rest of the year. For instance, if the energy consumed by periodic retraining during the available 6 months for Backblaze is 6.3K, we assume that in the following 6 months, the model will consume the same amount of energy.

In Table 4.2 we show the energy consumed by the most sustainable retraining technique for each dataset (KS-FI for Backblaze and Alibaba and KS-ALL for Google) vs. periodic retraining and an estimate of energy consumed over one year. Our results show that retraining using a suitable drift detector can significantly reduce energy consumption over one year. More specifically it can reduce the energy consumed by failure prediction models for Google and Alibaba by around 1000 kilojoules (around 40% less energy for Alibaba and 10% for Google) and the energy consumed by failure prediction models for Backblaze

by almost 1 kilojoule (around 7%). Therefore, employing a drift detector that is not too sensitive and can identify when a model requires retraining is substantially beneficial in the long term compared to periodic retraining.

RQ1c Answer

There is no one-size-fits-all solution for model retraining and different options should be considered before opting for a particular technique. Although there is evidence of the benefits of using an informed retraining technique, the main challenge lies in choosing an appropriate drift detector and in the frequency of concept drift occurrences. Furthermore, the optimal informed retraining technique should be paired with a sliding window retraining approach to minimize the consumed energy.



### 4.6.2 Impact on Inference Energy

Our study aims to understand whether different model retraining techniques can impact the energy consumed during inference. Therefore, in Fig. 4.3 we depict the distribution of the energy consumed during inference for each model across all random seeds. From Fig. 4.3 we can notice that the variation in inference energy consumption for each retraining technique is relatively small. Therefore, although each retraining technique is different in terms of the frequency of retraining a model and the data, the energy on inference is not impacted.

We verified whether there is a significant difference between the energy consumed during the inference for all analyzed scenarios (p-values higher than 0.05). For the Google dataset, there was no significant difference in energy consumed during inference between all the analyzed retraining techniques. For the Alibaba dataset, both periodic retraining techniques (Periodic_SW and Periodic_FH) are significantly different from the rest in terms of energy consumed during inference. However, the difference in consumed energy is only around 10 joules. When it comes to Backblaze, there was no significant difference in the energy consumed during inference between the Static and KS-ALL or between KS-PCA, KS-FI, and Periodic models. However, although for instance the Static and Periodic models consume significantly different amounts of energy according to the Wilcoxon statistical test (p-value 0.04), this difference was only around 1.5 joules. Thereby, the difference in energy consumed during inference between the retraining techniques is either not significant or extremely low.

RQ2 Answer

In general, the employed retraining technique does not affect the energy consumption of inference tasks.

## 4.7 Discussion and Implications

In this section, we will highlight the general findings derived from this study and we will discuss our results. Our findings aim to help ML practitioners build more sustainable

ML software systems and understand how to experimentally assess different retraining techniques in terms of sustainability. The remainder of this section will present each of our findings followed by a discussion and its implications.

**Finding 1**

Retraining using only the newest data is typically more sustainable than retraining on all available data and has a negligible influence on the model's performance.

In our work, we validate the claim of the authors [44] that employing a sliding window approach when retraining ML models is more sustainable with experimental evidence. When comparing the sliding window and the full-history approach we noticed that for all datasets the difference in model performance, ROC AUC, is minimal, while the difference in energy consumption is considerable. For the Backblaze dataset, the ROC AUC obtained by both retraining approaches is the same, while for the Alibaba dataset, the sliding window approach achieved an ROC AUC of 1% higher than the full-history approach, showing that the model accuracy benefits from deleting older samples. The only dataset where the full-history approach has a higher ROC AUC than the sliding window approach is the Google dataset. However, from all three datasets, the Google dataset is the shortest in terms of sample collection (29 days instead of 2 months for Alibaba and one year for Backblaze). Therefore, the reason why a full-history approach can be more beneficial from the accuracy perspective is that the model might require more training samples before deploying the model in production. Furthermore, the difference in energy consumed by retraining the Google dataset with a full-history approach vs. a sliding window approach is significantly high, ranging from 130 kilojoules to 222 kilojoules depending on the configuration, while the ROC AUC gain is only a maximum of 1.5%. Due to the minimal accuracy improvement and the high energy consumption associated with using a full-history approach, we recommend that ML practitioners adopt a sliding window retraining technique when developing Green ML applications.

**Finding 2**

Retraining a model based on a drift detector can benefit both the model's performance and the energy consumed **only** if the drift detector is properly chosen.

While answering RQ1b and RQ1c, we noticed that integrating a drift detector as an indicator of when to retrain can be beneficial only if we know that the detector is not too sensitive or the detector can identify drifts. We had an example of employing a drift detector that was not able to identify any drift for the Alibaba dataset, namely the KS-ALL. In this situation, using this drift detector leads to obtaining the same ROC AUC as not retraining the model at all (Static), while slightly increasing the overall energy consumption of the model in production, since the energy consumed by the drift detector needs to be considered. We also had an example of what happens if the drift detector is too sensitive and constantly signals drifts in the experiment with the Google dataset (drift detectors KS-PCA and KS-FI). Here we observed that if a drift detector is too sensitive, the amount of energy consumed by a configuration that retrains based on drift detection becomes

higher than the amount of energy consumed during periodic retraining, while sometimes (in the case of KS-FI) lowering the ROC AUC by 1%. However, for all three datasets, there was at least one configuration involving drift detection that both reduced the consumed training energy and preserved the ROC AUC. For this reason, we argue that a drift detector can benefit both the consumed energy and the model's performance only if it is properly chosen.

**Finding 3**

Although inference is not affected, the energy consumption of training a model is severely impacted by the retraining strategy.

Throughout our experiments, we noticed that the difference in consumed energy during inference among different retraining strategies is either not significant or extremely small. This shows that inference energy consumption is in our experimental setup not influenced by the employed retraining technique. However, we solely experimented with the Random Forest classifiers since this is the state of the art of failure prediction, which is the case study of this chapter. Therefore, our conclusions apply to this classification algorithm, but future work should investigate whether the same conclusions hold when employing other classification algorithms or when investigating other study cases, such as deep learning applications, which are more energy-intensive when it comes to inference.



**Finding 4**

The retraining frequency has a significant impact on the energy consumption.

While answering RQ1c, we performed a one-year estimation to compare the long-term benefits of employing drift-based retraining with periodic retraining. Throughout this experiment, we also noticed that the most energy-intensive model is the failure prediction model for the Google dataset (approx. 10 megajoules consumed during one year), followed by Alibaba (approx. 1 megajoules) and Backblaze (approx. 6 kilojoules). The main difference between these models in the energy consumed over one year is the retraining frequency, since Backblaze and Google models are retrained every month and the Alibaba dataset is retrained every week. Therefore, the retraining frequency has a significant impact on the model's energy consumption. However, the retraining frequency is context dependent because it must consider the business context and how the ML application is used in real world. Hence, given the tremendous impact of the retraining frequency on the energy consumed, we recommend ML practitioners to design ML applications that require retraining less often if the business context allows.

## 4.8 Threats to Validity

**External validity.** Our study focuses solely on Failure Prediction models, where concept drift is a known issue [17, 18, 37]. To support replication, we provide a replication package with all the code necessary to execute experiments. All employed datasets are publicly available (Backblaze, Google, Alibaba). We use Random Forests, the state-of-the-art for

this task, but results may vary with other algorithms. Due to time constraints, we limited our scope, though future work should explore other domains and models.

**Internal validity.** Our experiments replicate state-of-the-art literature on failure prediction models [17, 18, 37] and we acknowledge that different configurations can challenge the results we collect. Nevertheless, our replication package paves the work for future research to expand the scope of our subjects.

**Construct validity** We measured energy only for training, inference, and drift detection, as other pipeline parts should not affected by retraining methods. Due to a one-week experiment budget, we ran fewer repetitions for the Google dataset, but results remain valid since 30 seeds (on Alibaba and Backblaze) showed little variation compared to 5 (used for Google).



## 4.9 Conclusions and Future Work

In this study, we investigated the effects on training and inference energy consumption of multiple retraining techniques (sliding window vs. a full-history and periodic vs. informed). We conduct our experiments on a real-world ML application, namely failure prediction.

We provide empirical evidence that retraining a model only on the newest data (sliding window) is more sustainable than retraining a model on all available data (full-history) [29, 44]. Furthermore, employing a full-history approach does not always come with a benefit in terms of the models' performance compared to sliding windows. We further showed that retraining based on unsupervised drift detectors is better than retraining periodically w.r.t. energy consumption only when the drift detection technique is not too sensitive and is capable of identifying drift. To understand the long-term benefits of employing a drift detection-based retraining technique, we showed that using an appropriate drift detector is estimated to decrease energy consumption by up to 40% during one year compared to period retraining. Furthermore, we demonstrated that in most situations the retraining approach does not influence the energy consumed during inference.

The general conclusions derived from this study should serve as a guideline for ML practitioners to build Green ML applications. If the business context allows, practitioners should design ML applications that require less frequent retraining. For example, an ML application retrained monthly is more sustainable than retrained weekly. Practitioners should discard old data when retraining the model and should adopt a drift detection-based retraining approach to optimize energy efficiency. Unsupervised drift detectors that perform feature reduction (KS-FI and KS-PCA) are usually more sustainable than the ones that do not (KS-ALL). We recommend practitioners carefully choose the drift detector that is the most suitable for their datasets and analyze their consequences on the ML model's consumed energy and accuracy in the long term.

**Future Work:** Based on our findings, drift detection-based retraining can solely be beneficial in terms of both energy consumption and the model's accuracy if the drift detector is properly chosen. Therefore in the future, we plan to develop a framework that enables ML practitioners to check whether a drift detector is too sensitive to data changes or incapable of detecting drifts. In this chapter, we solely focused on the energy consumption

of AIOps applications since these applications are widely used in industry [21], they use plenty of data, and they are sensitive to concept drift [17]. We plan to expand this study to other energy-intensive applications where model update is also critical, such as large language models or other deep learning applications.

# 5

# Is Your Anomaly Detector Ready for Change? Adapting AIOps Solutions to the Real World



*Anomaly detection techniques are essential in automating the monitoring of IT systems and operations. These techniques imply that machine learning algorithms are trained on operational data corresponding to a specific period of time and that they are continuously evaluated on newly emerging data. Operational data is constantly changing over time, which affects the performance of deployed anomaly detection models. Therefore, continuous model maintenance is required to preserve the performance of anomaly detectors over time. In this work, we analyze two different anomaly detection model maintenance techniques in terms of the model update frequency, namely blind model retraining and informed model retraining. We further investigate the effects of updating the model by retraining it on all the available data (full-history approach) and only the newest data (sliding window approach). Moreover, we investigate whether a data change monitoring tool is capable of determining when the anomaly detection model needs to be updated through retraining.*

## 5.1 Introduction

The field of AIOps refers to applying artificial intelligence (AI) techniques on large-scale operational data to solve challenges derived from operational workflows. AIOps aims to

This chapter is based on the following peer-reviewed publication:

*Lorena Poenaru-Olaru, Natalia Karpova, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. 2024. Is Your Anomaly Detector Ready for Change? Adapting AIOps Solutions to the Real World. IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN '24), Melbourne, Australia, 2023, pp. 98-99* [47].

increase the productivity of IT Ops, DevOps, and software reliability engineering teams by predicting the behavior of large-scale software systems and improving the software architectural decision-making processes [20]. The term AIOps solutions is used to describe machine learning (ML) systems that learn from operational data. In the past years, AIOps solutions have witnessed a fast adoption within different industries. The most popular AIOps solutions are failure prediction and anomaly detection [111].

Given that any AIOps solution is an ML system, the quality of its predictions is strongly dependent on the data it was trained on and the data it is evaluated on after being deployed into production. If the training data is significantly different than the evaluation data, there is a high chance that the performance of the ML system will be affected [41]. Previous work on failure prediction AIOps solutions has observed the evolving character of operational data, which implies that this data is continuously changing over time [17], [112], [35], [36]. The continuous data changes, also known as concept drift, influence the performance of AIOps solutions during their lifecycle. They become less accurate over time, which tremendously affects their reliability among practitioners. Although the concept drift cannot be prevented since it is caused by external hidden factors, AIOps practitioners need to constantly ensure that their AIOps solutions are up to date.



One solution that ML practitioners adopt to handle the evolving character of data is retraining/updating ML models over time [113]. Periodical model retraining has also been studied for failure detection AIOps solutions [37], [17] and has proved that continuous model updates achieve better performance over time compared to non-updated models. However, the effects of continuously updated models have only been studied for failure prediction models [17], [37].

In [42] the authors proposed a solution to handle concept drift in failure prediction AIOps solutions, which implies that the model is updated only when the concept drift is detected, instead of periodically. Therefore, their solution contains a monitoring part responsible for identifying concept drift in the evaluation data using a concept drift detector. However, this framework was just proposed, but it was never assessed on any AIOps solution.

When it comes to periodic model retraining, this technique has been studied for classification problems such as failure predictions AIOps solutions. Lyu et al. [37] suggested that it also needs to be studied on other AIOps solutions, such as anomaly detection on time series data. By doing so, AIOps practitioners could have a better understanding of whether they can mitigate the effects of concept drift by constantly updating anomaly detectors. Regarding concept drift monitoring-based model retraining, this technique was not previously applied to any AIOps solution. Furthermore, previous work suggests that organizations do not have monitoring infrastructure to detect drift in production [113], [114] and they only perform periodic model retraining based on human decisions. Examining the impact of drift detection monitoring tools is the first step toward automating the maintenance of machine learning in production. Therefore, it could help in understanding whether concept drift detectors could be quality and reliability indicators for deployed ML models.

To mitigate the effects of concept drift on anomaly detection AIOps solutions, in this chapter, we study different model adaptation techniques. Our contributions can be summarized as follows:

1. We examine the effect of periodically updating models on the performance of anomaly

detection AIOps solutions.
2. We assess and report the limitations of the state-of-the-art anomaly detection models when being evaluated on different data sizes.
3. We investigate the effects of retraining anomaly detection AIOps models based on the output of a concept drift detector.
4. A publicly available replication package is provided including the implementation of a concept drift detector for time series.

## 5.2 Background and Related Work

### 5.2.1 Background AIOps Solutions & Anomaly Detection

AIOps solutions aim to identify issues in large software systems and then help with mitigating them or providing recommendations to engineers. When it comes to detecting possible issues in the system, a variety of different AIOps solutions were proposed for failure prediction tasks, namely predicting job failure [7], node failure [8], disk failure [6], incident [115] or outage [116]. Besides failures, plenty of attention has been paid to identifying abnormal system behavior, such as performance anomalies [117], [118], anomalies in system logs [119], [120], or internet traffic anomalies [121].



Although plenty of anomaly detection techniques were proposed in the literature, regarding AIOps solutions for anomaly detection, previous work has focused chiefly on *unsupervised* and *semi-supervised* models. This is a technique that practitioners use to handle the label availability challenge and high cost of obtaining true labels [121]. The best-performing and most popular techniques to detect anomalies in univariate AIOps data [122], [121], [123] belong to the *signal reconstruction models* group [124].

Anomalies are detected using methods that encode the time series into a latent space, such as Fast *Fourier Transform (FFT)* [125], *Spectral Residuals (SR)* [121] or *Prediction Confidence Interval (PCI)* [126]. These techniques usually have low computational costs, but they lose information during the encoding process [124]. To preserve more information, a type of artificial neural network called Auto-Encoder (AE) is employed to transform the time series data into a latent space. Thereby, plenty of anomaly detection techniques based on Auto-Encoders were derived, namely *Long Short-Term Memory Autoencoder (LSTM-AE)* [127] or *DONUT* [128]. Furthermore, Microsoft presents a more complex anomaly detector, *Spectral Residuals Convolutional Neural Networks (SR-CNN)* [121] that learns from multiple time series, generates synthetic anomalous samples, and trains a Convolutional Neural Network (CNN) to distinguish between anomalous and non-anomalous samples.

The anomaly detection models were previously evaluated using a delay metric. The reason for this is that in real-world applications, anomalies can occur either as single points or as segments of anomalies (group of continuous anomalies) [129]. According to [121], [130], [122], in AIOps solutions detecting any anomaly point in a segment of anomalies with a relatively small delay is considered as successful as detecting all anomaly points belonging to the same segment. Therefore, a delay-based evaluation strategy was presented in previous anomaly detection studies [121], [130], [122].

### 5.2.2 Concept Drift in Operational Data and Model Adaptation Techniques

Previous work observed the evolving character of operational data, which is responsible for changes over time and, therefore, for concept drift [17], [20], [35], [36]. The presence of concept drift results in the degradation in performance of the failure prediction models. Therefore, AIOps solutions need to be constantly maintained over time by continuous retraining [17], [37].

Gama et al. [15] propose two machine learning retraining techniques from the perspective of the retraining frequency, namely *blind retraining* and *informed retraining*. Blind retraining is the equivalent of periodic retraining, where the model is retrained after a predefined period. This technique was previously used by previous work on failure prediction models [35], [36], [17], [37], [131] and proved to be beneficial for preserving the model's performance over time. Informed retraining implies the existence of a data monitoring tool called a *concept drift detector* which indicates when the model is outdated due to changes in data. However, there is currently no study on the effects of blind and informed retraining on anomaly detection models.



Lyu et al. [37] propose two retraining techniques from the perspective of retraining data, namely the *full-history approach* and the *sliding window approach*. The full-history approach constantly enriches the training dataset with the newest data and retrains the model, while the sliding window approach retrains the model only on the most recent data, discarding old samples. These methods have been studied in failure prediction models, but there is currently no work on the effects of the full-history approach and sliding window on anomaly detection models.

### 5.2.3 Concept Drift Detection

Concept drift is monitored using some algorithms that can capture the moment when data shift occurs called *concept drift detectors* [41]. Although there are plenty of drift detectors available for multivariate data used in classification purposes when it comes to time series, the number of available drift detection techniques is significantly lower [14]. According to Bayram et al. [14], the reason for this is the lack of available open-source relevant time series datasets to study drift detection. Despite the potential of concept drift detectors to monitor data against concept drift, they have not been yet applied to the AIOps domain. Furthermore, there is no study on concept drift monitoring techniques for anomaly detection on operational data.

Out of the existing concept drift detectors for time series we can mention *Feature Extraction Drift Detection (FEDD)* [132] and *Entropy-Based Time Domain Feature Extraction (ETFE)* [133]. Both these detectors identify drift by extracting features from a given time series window and observing their similarity with the features extracted from the reference window. FEDD is extracting six linear (autocorrelation, partial autocorrelation, variance, skewness coefficient, kurtosis coefficient, and turning point rate) and two non-linear features (bicorrelation and mutual information) and computing the dissimilarity between features extracted in the current window and the reference window. The drift is detected through a change detector which analyses the exponentially weighted moving average (EWMA) of the computed dissimilarities. ETFE is extracting solely entropy-related features (approximate, fuzzy, sample, permutation, increment, and weighted permutation entropy)

from the decomposed time series. Concept drift is detected from the features extracted from the reference and the current time series using the GLR statistical test.

## 5.3 Motivational Example

In Figure Figure 5.1, we depict an example of two different time series related to internet traffic from the Yahoo dataset, a popular benchmark for AIOps data related to internet traffic [123], [121], [122]. In the upper plot, the behavior of operational data is significantly changing after a certain timestamp, while in the lower plot, it remains relatively constant. Thus, different time series have different behaviors over time.

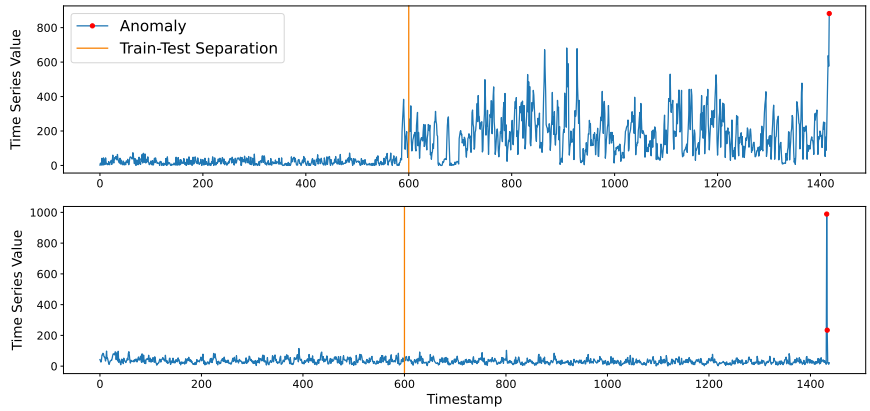



Figure 5.1: The time series to the left of the vertical line represents the training data, while the one to the right represents the evaluation/testing data.

We assume that we train two anomaly detection models on the data corresponding to the left of the vertical line and we keep the rest for testing. In the upper time series, we notice that the testing data is significantly different than the training data, which means that the model requires updating over time. Otherwise, the model would signal plenty of false alarms, which need to be verified by the operation and service engineers. Thus, the advantage of saving data monitoring time using anomaly detection AIOps solutions is diminished by the amount of time engineers need to spend searching for the causes of erroneously identified anomalies. Furthermore, the engineers would doubt the reliability of the model and they will be more reluctant to employ it. However, we notice that the data of the lower time series is not changing over time, thus the model should be able to capture anomalies.

If we would maintain our models using *blind retraining*, the model corresponding to the lower time series would be retrained although their performance might not be drastically improved since there were no changes in the data. Thus, we might encounter unnecessary retraining costs, which could be avoided by employing *informed retraining* where we initially verify whether data has changed and then update the model. When it comes to the model corresponding to the upper time series, both blind retraining and informed retraining might be beneficial.

Using a full-history approach in the case of the upper time series might result in poor anomaly detection performance for the upper time series since the past data no longer resembles current data. However, in the case of the lower time series, both techniques sliding window and full-history approaches might lead to similar results.

Given that time series are different, they might require different maintenance techniques. Therefore, in our study, we address the effects of updating different models over time on their performance and the implications of each maintenance technique in terms of how accurately they detect anomalies and how many false alarms they raise.

## 5.4 Research Questions

We begin by assessing the performance of state-of-the-art anomaly detection models on operational data. Moreover, we aim to understand whether the size of the testing set affects the performance of these models. We further aim to investigate the lifecycle of these models in the situation in which the models are maintained over time compared to when they are never updated. We analyze these aspects by evaluating the model retraining effects from the perspective of the retraining data and retraining frequency.



1. What is the performance of state-of-the-art anomaly detection models on operational data?
   (a) How robust are the state-of-the-art models to the testing set size?
2. What is the impact of the two model retraining techniques from the perspective of the retraining data (full-history vs. sliding window approach)?
3. What is the impact of the two model retraining techniques from the perspective of the retraining frequency (blind vs. informed retraining)?

## 5.5 Evaluation Methodology

In this section, we present the evaluation methodology that we employ to answer our research questions. We begin by describing the employed datasets and anomaly detection models. We continue by presenting different retraining (maintenance) techniques and the concept drift detector monitoring tool that signals changes in the data over time. Lastly, we explain the used evaluation metrics together with the delay tolerance.

### 5.5.1 Datasets

Although there are various benchmarks for anomaly detection, very few benchmarks for AIOps anomaly detection exist since research on AIOps is mostly performed using proprietary production data that is not publicly released [22]. Therefore, we selected two popular operational datasets containing univariate time series with different data collection granularity and lengths as presented in Table 5.1, that were previously used to build anomaly detection AIOps solutions, namely Yahoo S5[1] and the Numenta Anomaly Benchmark (NAB)[2]. Although the publicly available benchmarks received criticism in the way they are labeled [134], labeling anomalies in real-world AIOps datasets is complex, labor-intensive, and prone to error due to manual labeling and the subjectivity of the annotator [22]. For instance, some annotators correlate anomalies with the moment an

[1]Yahoo Data Source
[2]NAB Data Source

incident occurs [22], without taking into account that the points before and after the incident are identical as also pointed out by [134]. Furthermore, currently, the AIOps time series data are collected from various sources, which is the reason why anomalies exhibit different behaviors and they are not consistent across time-series [22]. Therefore, the anomaly detection AIOps datasets benchmarks reflect real-world characteristics of operational data, but as mentioned in [22], [134], the labeling process requires more transparency by understanding the human decision-making process while labeling and the cause of the anomaly.

Table 5.1: Overview of the datasets characteristics.

| Dataset | NAB | Yahoo A1 |
|---|---|---|
| Number of Time Series in Dataset | 17 | 67 |
| Granularity | 5 min | 1 hour |
| Approx. Min Data Collection Time | 5 days | 31 days |
| Approx. Max Data Collection Time | 17 days | 61 days |



**Yahoo** is a dataset released by Yahoo Lab which contains both synthetic and real data collected from *internet traffic* generated by Yahoo services. In our experiments, we solely considered real-world data (the Yahoo A1 benchmark), which we are further referring to as Yahoo. This dataset is composed of 67 time series collected with a granularity of 1 hour for 31 days (the shortest time series) and 61 days (the longest time series). The anomalous and non-anomalous points are labeled by domain experts.

**NAB** is a publicly available dataset previously used as a benchmark to assess the performance of anomaly detection models [135]. The NAB corpus contains multiple types of time series, such as metro traffic, tweets, etc. For our experiments, we solely select datasets referring to operational data, namely the realAWSCloudwatch time series that contain real-world server metrics (e.g. CPU utilization, Network Bytes In, Disk Read Bytes) collected by the AmazonCloudwatch service. For simplicity, we are further referring to these time series as NAB. This dataset is composed of 17 time series collected with a granularity of 5 minutes for 5 days (the shortest time series) and 17 days (the longest time series). To obtain the labels, we employ the ground truth labeling approach (labeling only the ground truth as an anomaly) instead of labeling the entire region to avoid raising too many false alarms to engineers [136].

**Data-Splitting And Preprocessing** In our study, we considered the same data-splitting scenario from previous works [121], [122], namely for each time series from each dataset the first half is used for training and the second half is used for testing. For our experiments, we assume that the anomaly labels are known for the training set.

### 5.5.2 Models

#### State-of-the-Art Anomaly Detection

We replicate five popular unsupervised anomaly detection models based on signal reconstruction, namely *FFT* [125], *SR* [121], *PCI* [126], *LSTM-AE* [127], and *SR-CNN* [121]. We

selected our models based on both their popularity and their dissimilarity in techniques to detect anomalies. Furthermore, these models were previously used for anomaly detection in AIOps [121], [122], [123]. Therefore, we employ three models that work by solely applying different mathematical operations of the given time series to detect anomalies (FFT, PCI, and SR) and two models that require learning the behavior of the time series (LSTM-AE, and SR-CNN).

The **FFT** method firstly transforms the time series from the time domain into the frequency domain using the Fourier transform and then transforms the time series back into the time domain to estimate a fitted curve of the data. Anomalies are detected by finding differences between data points and the fitted curves. In our study, we use the implementation of FFT from the original paper [125], which is publicly available[3]. Given the unavailability of a threshold that differentiates anomalies from non-anomalies, we empirically computed the threshold using the training data. Therefore, we remove all existing anomalies from the training data and we extract the maximum anomaly score on the training data. Sample with a higher anomaly score than the threshold are classified as anomalies.



The **SR** method is composed of three steps: the Fast Fourier Transform, which transforms the time series from the time domain into the frequency domain to calculate the log amplitude spectrum, the spectral residuals calculation and the Inverse Fourier Transform transforms the signal from the frequency domain back into the time domain. After these three steps, the saliency map is computed and used to detect anomalies through a threshold. In our study, we use the publicly available implementation of SR[4] from the original paper [121]. Optimal hyperparameters are determined by performing a grid search starting from the parameters suggested in [121] until similar results are obtained on Yahoo. The NAB dataset is not assessed in the original paper and, thus, we perform a grid search to identify the optimal parameters.

The **PCI** method relies on the k-nearest neighbors of one data point to identify if that specific data point is an anomaly. The k-nearest neighbors are the closest continuous data points to the targeted data point in the time series. After calculating the nearest neighbors, the prediction confidence interval is calculated. The current data point is classified as an anomaly if its value is outside of the prediction confidence interval. We use the publicly available implementation of PCI[5] from the original paper [126]. Optimal parameters are computed for each dataset through grid search.

The **LSTM-AE** method uses an autoencoder architecture to learn the representation of a time series in the time domain. An anomaly is detected when the error between the reconstructed and true time series is higher than a predefined threshold. Due to the lack of publicly available implementations of LSTM-AE, we implement it based on the architecture presented in [137]. We remove anomalous samples from the training set [122] and perform interpolation between the closest non-anomalous samples is performed to preserve the continuity of the time series. Our implementation is publicly available in our replication package[6].

[3]FFT Open Source Implementation
[4]SR Open Source Implementation
[5]PCI Open Source Implementation
[6]Replication Package

The **SR-CNN** method is composed of two main parts: the synthetic data generation and the CNN. It uses the saliency map calculated through spectral residuals as features for the CNN. The synthetic data generator injects fake anomalous points into the saliency map to reduce the high imbalance between anomalous and non-anomalous points. The augmented data is used to train the CNN and distinguish between anomalies and non-anomalies. We used the publicly available implementation[7] of SR-CNN from the original paper [121]. We perform a grid search to determine the optimal parameters of SR-CNN on both datasets.

#### Testing Window Size Assessment

Previous work reported results by splitting the original time series into half and using the entire first half for training and the entire second half for testing. We are further referring to this experimental setup as the SoTA setup. However, since we aim to understand the effect of retraining techniques on the available datasets, we also need to ensure that the model's predictions are not impacted by a smaller testing window. Thus, we split the testing sets into smaller subsets, and we evaluate the same model on each of the subsets. We further combine the predictions on the subsets and compare them to the prediction obtained from the SoTA setup. We are further referring to this setup as the Window Size Setup. The testing subsets correspond to around 168 samples (one week) for Yahoo and 225 samples (one day) for NAB and they also represent the retraining periods. We use natural time intervals, one week and one day for Yahoo and NAB, respectively, since they are commonly used in model retraining for AIOps solutions [17], [35], [36], [107]. Moreover, these exact periods were chosen based on the amount of available data points in each time series (see Table 5.1). The same subsets are used in the model retraining experiments.



### 5.5.3 Retraining Techniques

#### Retraining Data

From a training data perspective, we compare three scenarios, namely the scenario when the model is never updated, which we are referring to as the *static approach* and two scenarios in which the model is updated over time, namely the *full history approach* and *sliding window approach*.

The testing data is split into equal testing batches corresponding to the predefined period. The experimental setup for the 3 scenarios can be observed in Figure Figure 5.2. According to each scenario, the model update is done as follows:

**Static Approach (S):** The model is trained only once on the training data and tested on the testing batches.

**Full History Approach (FH):** The model is retrained periodically by constantly enriching the training set. Thus, when testing the model on the testing set corresponding to batch *t*, the model is trained on all the available data until batch *t-1*.

**Sliding Window Approach (SW):** The model is retrained periodically by constantly replacing old data with new data. Thus, when testing the model on the testing set corresponding to batch *t*, the model is retrained on only the newest data. In this approach, the training set size remains constant over time.

[7] SR-CNN Open Source Implementation

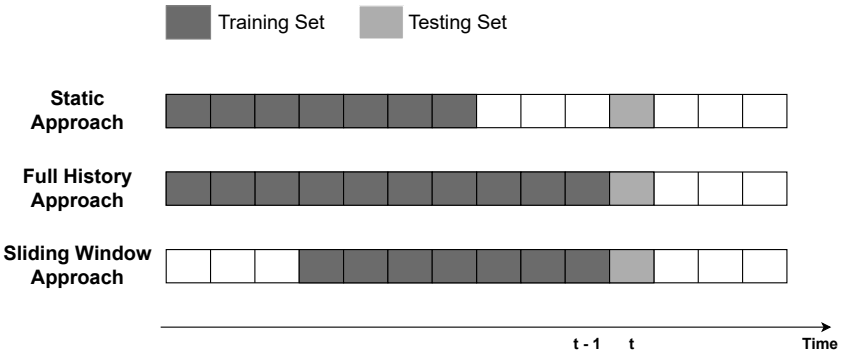



Figure 5.2: Training and testing data in case of the static approach, full-history approach, and sliding window approach.



#### Retraining Frequency

From a retraining frequency perspective, we compare *blind retraining* with *informed retraining* updated models. The blind retraining technique implies that the model is retrained on a periodic basis by constantly including testing batches in the training data. Informed retraining implies that the model is retrained on all the data including the batch when the change occurs, only when a change detector identifies changes in data. A visual representation of the two is given in Figure Figure 5.3.

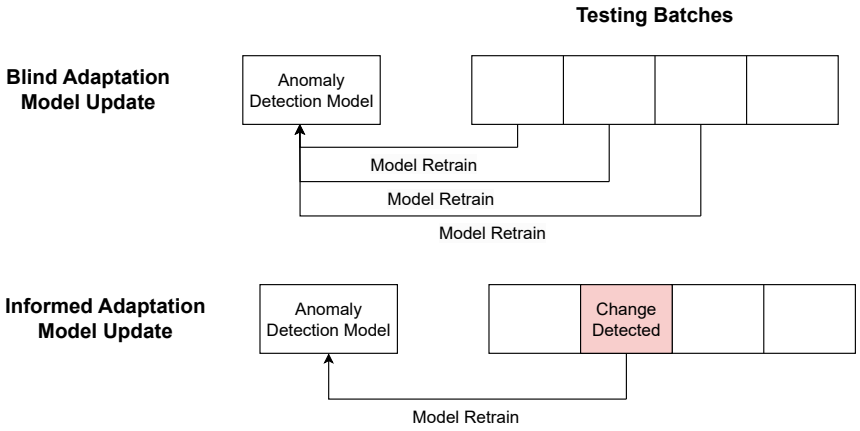



Figure 5.3: Blind vs. Informed Retraining.

### 5.5.4 Concept Drift Detectors

For this study, we only consider FEDD as a concept drift detector since it was evaluated on both synthetic and real-world data [138]. For our experiments, we extract the im-

plementation of FEDD from the replication package[8] provided by the authors of [138]. Furthermore, we translate the code into English and include the FEDD implementation in our replication package together with an example of how to run it on any time series such that practitioners can use it for their data.

### 5.5.5 Model Evaluation

#### Evaluation Metrics

We use F1-score, Precision, and Recall to evaluate the performance of anomaly detectors. We do not consider the computation time as one of our evaluation metrics since the evaluated time series are relatively short and, thus, meaningful conclusions cannot be drawn. The Precision metric shows the ability of the anomaly detector not to label a non-anomaly as an anomaly and, therefore, not to raise too many false alarms. The Recall metric shows the ability of the anomaly detector to find all the anomalies and, therefore, to correctly identify all the anomalies. The F1-Score shows the compromise between Precision and Recall and, therefore, how many anomalies are correctly identified with respect to how many false alarms are triggered. The metrics are defined by the following equations:



$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

where TP and FN represent the number of True Positives and False Negatives, respectively.

#### Delay Metric

As aforementioned, AIOps anomaly detection models were evaluated considering a predefined prediction tolerance called delay. The detection delay is the maximum tolerable number of anomalous points in the anomaly segment that the detector can omit. If the detectors find any anomaly in the segment within this delay, the detection is considered correct. Otherwise, the detector fails to capture the anomaly. We depict in Figure Figure 5.4 an example of metric computation including the delay.

#### Statistical Test

We report the metrics averaged over all time series belonging to one dataset as done in previous work [121], [130], [122]. To understand whether the differences between the averaged metrics are statistically significant we employ the Wilcoxon signed-rank statistical test and assess whether the difference between the metrics obtained for each time series in one scenario and the metrics obtained for each time series in another scenario is statistically significant. The Wilcoxon signed rank is a non-parametric statistical test, thus it does not make any assumption about the data distributions. Its null hypothesis is that the two populations analyzed come from the same distribution and, thus, there is no

[8] FEDD Open Source Implementation

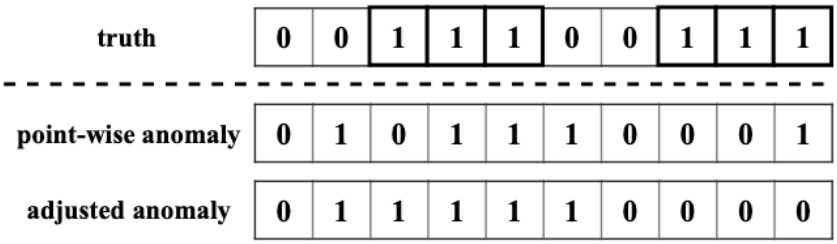



Figure 5.4: In this example we show the labels for 10 data points corresponding to one time series, where 1 indicates an anomaly and 0 indicates a non-anomalous point. The first row shows the ground truth, the second row shows the original predictions of one anomaly detection model and the third row shows the adjusted predictions considering a delay of 1. In the ground truth, there are 2 anomalous segments, each containing 3 anomalies. Since the model managed to predict the second anomaly in the sequence of anomalies and we tolerate a delay of 1 sample, the adjusted anomaly treats the entire first anomaly segment as a correct prediction. However, the second sequence of anomalies is treated as an incorrect prediction since even with a delay of one the anomaly on position 8 is not reported in time, while with a delay of 2, it would be reported in time.[121]



statistically significant difference between the two scenarios. We use a confidence interval of 90% to assess whether the null hypothesis is accepted or rejected. To avoid bias in our experiments, we employ 5 random seeds in the case of LSTM-AE and SR-CNN, which suffer from randomness. We apply the statistical test on each random seed to assess the significance.

## 5.6 Experimental Results

### 5.6.1 Anomaly Detection Models on Operational Data

#### State-of-the-Art Setup

The first set of experiments aims to assess the performance of state-of-the-art anomaly detection models on operational data. We initially train the model on the first half of each time series and test it on the second half (SoTA setup), similar to the state-of-the-art [121], [130], [122].

We depict our findings in Table 5.2 where we can observe that the performance of anomaly detection models is overall relatively low, which could be a reason for the labeling problems identified in [134]. FFT and PCI models obtained the lowest F1-score for both datasets. When it comes to PCI, we can observe that it tends to classify most data points as anomalies given its high recall and low precision on both datasets. The same tendency can be observed when it comes to FFT on the Yahoo dataset and the SR on the NAB dataset. Given the low performance of FFT and PCI on both operational datasets, these two models are no longer considered in the following experiments. Moreover, the SR model is no longer considered in the following experiments on the NAB dataset for the same reason. The best-performing models on both datasets are LSTM-AE and SR-CNN.

In Table 5.2 we show the results of each model when performing a strict evaluation, where no delay is tolerated (delay=0). Thus, the model is penalized when it does not manage to detect the anomaly at the exact moment when it occurs. As aforementioned,

Table 5.2: Results of anomaly detectors in SoTA setting. With **bold** we highlight the best scores. The obtained results are obtained by averaging the metric over all time series from each dataset and over the 5 random seeds.

| | Model | F1-Score | Precision | Recall |
|---|---|---|---|---|
| Yahoo | FFT | 0.07 | 0.04 | 0.39 |
| | SR | **0.25** | **0.28** | **0.36** |
| | PCI | 0.09 | 0.06 | 0.39 |
| | LSTM-AE | **0.37** | **0.38** | **0.49** |
| | SR-CNN | **0.57** | **0.54** | **0.61** |
| NAB | FFT | 0.04 | 0.02 | 0.12 |
| | SR | 0.05 | 0.15 | 0.33 |
| | PCI | 0.03 | 0.02 | 0.32 |
| | LSTM-AE | **0.31** | **0.37** | **0.44** |
| | SR-CNN | **0.82** | **0.70** | **0.99** |

AIOps models were evaluated in previous studies [123], [121], [122] with a maximum delay of 7 points. Therefore, in Figure Figure 5.5 we show the results for the same models with a delay of up to 7 points.



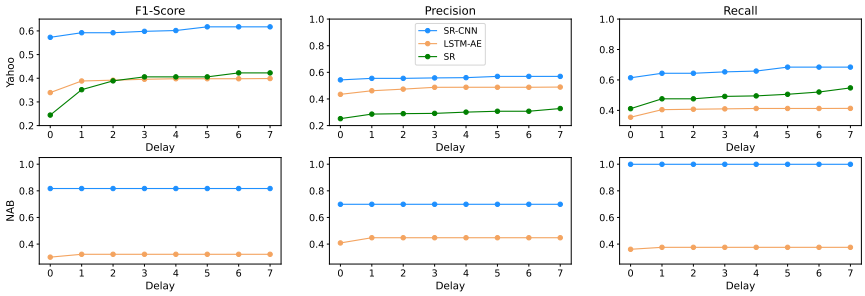



Figure 5.5: Delay metric applied to F1-score, precision and recall on Yahoo and NAB.

From Figure 5.5 we can notice that although on the Yahoo dataset, a higher delay can significantly influence the performance of anomaly detection models, on the NAB dataset it has almost no impact. In the case of Yahoo, the explanation behind this result is that 73.13% of the time series belonging to the Yahoo dataset contain sequences of anomalies (group anomalies). When it comes to NAB, only 29.41% of the time series contain sequences of anomalies. Therefore, the dataset containing samples related to internet traffic (Yahoo) contains significantly more group anomalies compared to the dataset containing samples related to CPU utilization (NAB), which shows the different behavior of different operational datasets.

Another important observation is that in the case of Yahoo, the SR model performs better after a delay of 2 points compared to LSTM-AE. However, if we assess the two anomaly detection models according to how well they detect the exact moment anomalies occur, the difference in the F1-score is almost 10%. In our work, we aim to understand how well these methods perform when detecting the exact anomalous point is crucial.

Therefore, for the rest of our experiments, we are solely considering a delay of 0 points when evaluating different anomaly detection models.

#### Window Size Setup

To fully answer our first research question, we need to understand whether models are robust towards the testing data size. We, therefore, employ the Window Size Setup described in Section 5.2.2 in the case of SR-CNN and LSTM-AE and compare it to the performance of the model when using the SoTA setting. However, as part of the functionality of SR, this model requires to be evaluated on a continuous time series. Therefore, this model could only be assessed for robustness on the sub-testing set immediately following the training set. From this experiment, we noticed that SR is not influenced by the size of the testing lengths and, thus, is robust to changes in the testing length. For SR-CNN and LSTM-AE we show results in Table 5.3.

Table 5.3: Assessment of anomaly detectors' robustness towards testing window size (Window Size Setup). The obtained results are obtained by averaging the metric over all time series included in one dataset and over the 5 random seeds.



| | Model | F1-Score | Precision | Recall |
|---|---|---|---|---|
| **Yahoo** | LSTM-AE | 0.37 | 0.38 | 0.49 |
| | SR-CNN | 0.14 | 0.14 | 0.14 |
| **NAB** | LSTM-AE | 0.31 | 0.37 | 0.44 |
| | SR-CNN | 0.05 | 0.05 | 0.05 |

Throughout our experiments, we noticed that the performance of LSTM-AE in both scenarios is the same. Therefore, the testing window size does not influence this anomaly detector. As it can be observed from Table 5.3, SR-CNN is drastically affected by being evaluated on smaller window sizes on both datasets. Its performance drops from an F1-score of 0.82 to 0.05 on the NAB dataset and from an F1-score of 0.57 to 0.14 for the Yahoo dataset.Thus, we consider that only SR and LSTM-AE are robust toward different testing sizes.

### 5.6.2 Full-History vs. Sliding Window

In this set of experiments, we aim to answer our second research question and understand the differences in performance between the model that was never updated (S) compared to the models that were updated using a full-history (FH) vs. a sliding window (SW) approach. In Table 5.4 we show the results of this experiment. The results displayed for SR-CNN are the results obtained through the testing window size setup experiment since this experiment implies a division of the entire testing data into smaller subsets and retraining the model accordingly.

From Table 5.4 we can notice that the performance of LSTM-AE is significantly improving when the model is updated over time. The highest performance is achieved when the model is updated using the sliding window (SW) technique. When it comes to the SR model, its performance degrades when being updated with the SW technique, but it does not statistically change when being updated with the FH technique. The SR-CNN performance benefits from being updated using the FH technique since it manages to find

Table 5.4: Results of anomaly detectors for blind model retraining. With **bold** we highlight the situation in which the difference in performance between the S model and the updated model (FH, SW) is significant. The displayed results are obtained by averaging the metric over all time series included in one dataset and over the 5 random seeds. (S-Static, FH-Full-History, SW-Sliding Window)

| | | F1-Score | | | Precision | | | Recall | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Model** | **S** | **FH** | **SW** | **S** | **FH** | **SW** | **S** | **FH** | **SW** |
| **Yahoo** | SR | 0.25 | 0.21 | **0.19** | 0.28 | **0.19** | **0.17** | 0.36 | **0.23** | **0.23** |
| | LSTM-AE | 0.37 | **0.42** | **0.46** | 0.38 | **0.42** | **0.46** | 0.49 | **0.53** | **0.59** |
| | SR-CNN | 0.14 | 0.10 | 0.14 | 0.14 | 0.09 | 0.17 | 0.14 | **0.26** | 0.22 |
| **NAB** | LSTM-AE | 0.31 | **0.34** | **0.37** | 0.37 | 0.37 | **0.45** | 0.44 | **0.46** | **0.48** |
| | SR-CNN | 0.05 | 0.07 | 0.03 | 0.05 | **0.15** | 0.07 | 0.05 | **0.13** | 0.04 |

more anomalies (higher recall) and also to label more non-anomalies correctly (higher precision). In terms of SR, its performance degrades when old data is discarded.

### 5.6.3 Drift Detection based Retraining

#### Drift Detection Results

We apply the concept drift detector FEDD on a period basis (weekly for Yahoo and daily for NAB) for each time series from each dataset. Figure Figure 5.6 shows the percentage of time series affected by concept drift according to FEDD during each period. Due to the lengths of time series corresponding to each dataset, the evaluation time series within the Yahoo dataset could be split into a maximum of five periods (five weeks), while those within the NAB dataset could be split into a maximum of nine periods (nine days).

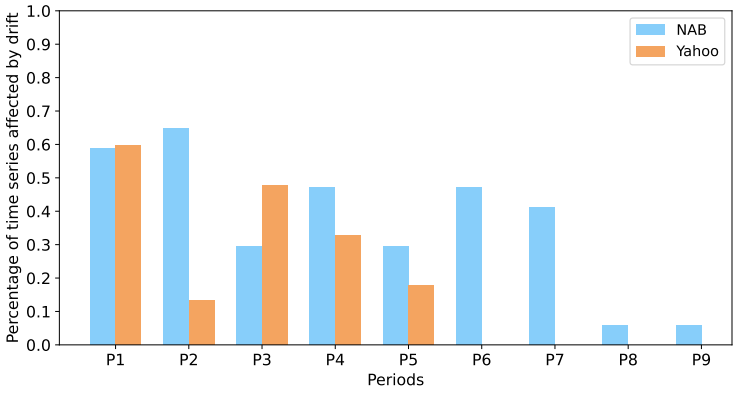



Figure 5.6: Percentage of time series affected by concept drift during each period according to FEDD.

From Figure Figure 5.6 we can see that both for Yahoo and NAB, more than 50% of the time series are affected by concept drift in the first period (P1). Furthermore, in the case of NAB, during the second period (P2), there are more than 60% of the time series affected by concept drift according to FEDD.

#### Blind vs. Informed Retraining

With this experiment, we aim to understand whether an anomaly detection model's performance can be preserved when retraining based on a drift detector's output (informed retraining) and how it compares to periodically retraining the model (blind retraining). Since the most significant benefits of periodic model retraining were observed in the case of LSTM-AE, this experiment is solely performed on this anomaly detection model. In addition, this experiment is performed with both data retraining techniques, full history (FH) and sliding window (SW).

Table 5.5 shows that overall the performance of informed model retraining is higher than the performance of a model that was never updated (static model). However, blind (periodic) model retraining generally achieves better results than informed model retraining. Similar to the periodic model retraining experiment, the highest results were achieved using the sliding window approach.

Table 5.5: Results of LSTM-AE when comparing blind with informed model retraining. The displayed results are obtained by averaging the metric over all time series included in one dataset and over the 5 random seeds. **Bold** means that informed retraining achieves significantly better results compared with the static model. *Italic* means that informed retraining achieves significantly similar results with blind retraining. (S-Static, FH-B/I-Full History Blind/Informed, SW-B/I-Sliding Window Blind/Informed.)

| | | S | FH-B | FH-I | SW-B | SW-I |
|---|---|---|---|---|---|---|
| **Yahoo** | F1 | 0.37 | 0.42 | **0.40** | 0.46 | **0.41** |
| | Precision | 0.38 | 0.42 | **0.40** | 0.46 | ***0.45*** |
| | Recall | 0.49 | 0.53 | 0.50 | 0.59 | **0.55** |
| **NAB** | F1 | 0.31 | 0.34 | ***0.34*** | 0.37 | ***0.36*** |
| | Precision | 0.37 | 0.37 | **0.39** | 0.45 | ***0.45*** |
| | Recall | 0.44 | 0.46 | 0.45 | 0.48 | **0.46** |

From Table 5.5 it can be noticed that the informed model retraining achieved similar precision to the blind model retraining for both datasets when the model was retrained using the sliding window approach. However, in all situations informed retraining results in lower recall compared with blind model retraining. When it comes to the full-history approach retraining technique, all performance metrics calculated for the blind model retraining are always higher than the performance metrics calculated for the informed model retraining on the Yahoo dataset. However, for the NAB dataset, the F1 score for informed model retraining is similar to the blind model retraining, while the precision is slightly higher.

## 5.7 Discussions and Answers to Research Questions

***Research Question 1:*** What is the performance of state-of-the-art anomaly detection models on operational data and how does the testing window size influence it?

***Answer 1:*** The more complex models (LSTM-AE and SR-CNN) perform significantly better on both operational datasets compared to the simpler models (FFT, PCI,

SR). However, the evaluation window size significantly impacts SR-CNN but does not affect LSTM-AE or SR.

**State-of-the-Art Anomaly Detection:** Throughout our experiments, we noticed that PCI and FFT tended to label every point in the time series as an anomaly. This shows the complexity of operational data also described in [22], which was only captured by LSTM-AE and SR-CNN. Moreover, we noticed a significant difference of almost 10% between LSTM-AE and SR anomaly detectors when the delay metric is considered. We believe that the delay could be misleading for practitioners who want to use the anomaly detection models and, thus we recommend researchers to initially report the results with a delay of 0 and afterward perform an additional experiment with a variable delay.

One interesting finding of our experiment is that SR-CNN is highly influenced by the size of the data it is evaluated on. When performing the experiments to assess this aspect, we noticed a significant decrease in performance when comparing the SR-CNN tested on the original setup described in the paper [121] to a setting in which the testing set is smaller. However, in both settings, we respected the original SR-CNN architecture in which the training and testing datasets are separated before performing the time series spectral transformation. This could be due to the functionality of SR-CNN since this model relies on decomposing time series into another domain and extracting features that are further used to train a CNN. Therefore, when shortening the time series, these features might not be significant enough for CNN to properly distinguish between anomalies and non-anomalies.



**Implications for Practitioners:** In the light of our findings we strongly recommend practitioners that when performing anomaly detection model selection they should consider a delay of 0. Thereafter, they can adjust the delay according to the requirements of their application. Given our findings regarding the performance drop of SR-CNN when a smaller testing size is employed, we further suggest that during model selection, AIOps practitioners should consider its robustness to different time series lengths and how often the inference is going to be performed. For instance, when identifying anomalies in a time series with a big granularity (e.g. 1 hour) that needs to be evaluated daily, employing an anomaly detector that is sensitive to short testing sets could lead to erroneous results. Moreover, sometimes incidents could occur and data collection could be interrupted, which can also shorten the available testing data.

***Research Question 2:*** What is the impact of the two model retraining techniques from the perspective of the retraining data (full-history vs. sliding window approach)?

***Answer 2:*** The sliding-window approach is beneficial to models that learn from the time domain (LSTM-AE) but can impact the performance of models decompose the time series into another domain (SR and SR-CNN). Retraining using the full-history approach achieved higher performance than retraining in most of the cases.

We empirically prove that the LSTM-AE anomaly detector benefits from being periodically updated in both datasets, being able to identify more anomalies and distinguish

better between anomalous and non-anomalous points. The performance of this model is higher when employing the sliding window approach than the full-history approach. This could show that old samples are no longer relevant to the current behavior of the data and retraining the LSTM-AE only on the most recent data eliminates non-relevant data points. When it comes to SR, which is not affected by the window size as shown in previous experiments, we noticed that a sliding window approach significantly impacts its performance. This can be a consequence of lowering the size of the time series and discarding important spectral information.

**Implications for Practitioners:** Based on our findings, periodically retraining anomaly detection models can significantly improve their performance over time. Thereby, AIOps practitioners should consider periodic model maintenance after deployment as part of the anomaly detection model lifecycle. Furthermore, we noticed that models that identify anomalies from the original time series (LSTM-AE) benefit from a sliding-window retraining approach, while the ones that detect anomalies by transforming the time series in other domains (SR) benefit from a full-history approach. Therefore, the retraining technique should be chosen according to the employed anomaly detector.



***Research Question 3:*** What is the impact of the two model retraining techniques from the perspective of the retraining frequency (blind vs. informed retraining)?

***Answer 3:*** Blind (periodic) retraining usually achieves higher performance than informed retraining. However, when a sliding window retraining technique is employed, the precision is statistically similar.

This experiment shows that retraining a model based on the output of a drift detector (FEDD) achieves a higher performance than a model that was never updated. This shows that the informed retraining setting is a promising research path and the FEDD drift detector could be employed as an anomaly detection model degradation monitoring tool. However, a model that is retrained periodically achieves slightly higher performance than a model retrained based on the output of a drift detector. This could be caused by the fact that not all drifts are captured by FEDD, which leaves room for improvement of concept drift detectors for time series.

We noticed from our experiments that for both datasets the precision obtained through blind retraining is comparable to the one obtained through informed retraining when a sliding window approach is employed. This shows that retraining based on a drift detector helps reduce the number of false alarms triggered by an anomaly detector as much as periodic retraining does. However, with a periodic model retraining more anomalies are captured compared to the blind model retraining given the higher recall.

**Implications for Practitioners:** Since we show the potential of having a model retrained based on the output of a drift detector, we argue that these monitoring tools can be employed in real-world anomaly detection model maintenance pipelines. Furthermore, to understand the suitability of the employed drift detector and the effects of retraining based on its output, we suggest that practitioners allocate a period to evaluate the model quality over time. Within this period AIOps practitioners should constantly investigate the effects of different retraining techniques (blind vs. informed and sliding-window vs. full-history) and understand which maintenance technique is the most suitable for anomaly

detection AIOps solutions with respect to its corresponding costs (labeling/labor costs, etc). Within this period multiple drift detectors can be evaluated .

## 5.8 Threats To Validity

### 5.8.1 External Validity

An external threat to validity derived from our study is the generalizability of our results towards different AIOps solutions. In our study, we targeted detecting anomalies in solely two AIOps domains, namely, internet traffic and CPU utilization due to public availability. Moreover, in this study, we relied on the provided anomalous labels corresponding to each dataset. The anomaly detection models depend on the datasets they are applied to. Although our datasets contain solely real-world data, the data used to e.g. predict incidents using AIOps solutions might be different than the one we employed. Therefore, the anomaly detection AIOps solutions lifecycle should be explored for other AIOps domains.

In our study, we included a variety of popular anomaly detection models, which are different from each other in terms of the technique they use to detect anomalies. Since we noticed that maintenance techniques are model-dependent when employing other models they need to be verified accordingly.



### 5.8.2 Internal Validity

When computing the static model, we used the first half of each time series from each dataset as training and the second half of each time series as testing as previous work [121], [122].

We partitioned our testing data into periods of different sizes (weekly and daily) similar to prior work [17], [37]. We did not consider periods smaller than one day since not enough data is captured in a time shorter than one day. Periods bigger than one week were not considered given the size of the time series.

### 5.8.3 Construct Validity

We replicated anomaly detection models used in previous studies [126], [121], [125]. We employed hyperparameters used by previous studies. In the case when the hyperparameters were not provided, we used hyperparameters that led to similar results to what was reported in previous studies. In the case of LSTM-AE, we used the maximum reconstruction error on the training set as our anomaly threshold, which is derived from each time series. We employed multiple performance metrics to evaluate and interpret the model's outcome.

## 5.9 Conclusions and Future Work

In this chapter, we investigate different maintenance techniques for popular anomaly detection models on two AIOps domains, internet traffic and CPU utilization. From a retraining frequency perspective, we analyzed *blind retraining*, retraining the model periodically, and *informed retraining*, retraining the model when a drift detector indicates it. With this study, we assess the potential of using drift detectors as a quality monitoring tool for anomaly detection models. From a retraining data perspective, we experimented with *full-history*

*approach*, constantly enriching the training set, and *sliding window approach*, discarding old samples from the training set.

We began our study by replicating the most popular anomaly detection model and testing their performances on the two AIOps datasets. We observed that some models could not detect anomalies in this type of data, while others were sensitive to the length of the testing data size. Our study shows that generally, an updated anomaly detector achieves higher performance than an anomaly detector that was never updated. We observed that the sliding-window approach benefits models that identify anomalies from the original time series while the full-history approach benefits models that transform time series into another domain to detect anomalies. Furthermore, we empirically demonstrated that the performance of an anomaly detector retrained based on the output of a drift detector leads to better performance when compared to an anomaly detector that was never updated. This shows that drift detection-based model update is a promising research path and can be the beginning of automated model maintenance pipelines. In light of our findings, we offer recommendations to AIOps practitioners regarding anomaly detection model selection and assessment, as well as identifying the most appropriate model maintenance techniques.



Our work was limited by the availability of open-source AIOps datasets for time series anomaly detection. We, therefore strongly encourage AIOps practitioners to release more datasets that researchers can use as benchmarks when developing new models. Moreover, we also believe that future research should focus on understanding and reporting the labeling process of experts. This can offer anomaly detection researchers more transparency and confidence in the data quality.

**Future Work:** In our study, we solely targeted anomaly detection for CPU utilization and internet traffic data. A promising research path is analyzing the potential of a concept drift detection-based framework in other AIOps applications, such as job and disk failure prediction [17]. Furthermore, we noticed that the performance of retraining an anomaly detector based on FEDD is lower than periodically updating the anomaly detector. This could show that FEDD does not capture all drifts that lead to performance degradation. Therefore, in the future we consider investigating more drift detectors suitable for time series beyond FEDD and designing more accurate drift detectors for time series.

# 6

# Prepared for the Unknown: Adapting AIOps Capacity Forecasting Models to Data Changes



*Capacity management is critical for software organizations to allocate resources effectively and meet operational demands. An important step in capacity management is predicting future resource needs often relies on data-driven analytics and machine learning (ML) forecasting models, which require frequent retraining to stay relevant as data evolves. Continuously retraining the forecasting models can be expensive and difficult to scale, posing a challenge for engineering teams tasked with balancing accuracy and efficiency. Retraining only when the data changes appears to be a more computationally efficient alternative, but its impact on accuracy requires further investigation. In this work, we investigate the effects of retraining capacity forecasting models for time series based on detected changes in the data compared to periodic retraining. Our results show that drift-based retraining achieves comparable forecasting accuracy to periodic retraining in most cases, making it a cost-effective strategy. However, in cases where data is changing rapidly, periodic retraining is still preferred to maximize the forecasting accuracy. These findings offer actionable insights for software teams to enhance forecasting systems, reducing retraining overhead while maintaining robust performance.*

This chapter is based on the following peer-reviewed publication:

*Lorena Poenaru-Olaru, Wouter van 't Hof, Adrian Stańdo, Arkadiusz P. Trawiński, Eileen Kapel, Jan S. Rellermeyer, Luis Cruz and Arie van Deursen. Prepared for the Unknown: Adapting AIOps Capacity Forecasting Models to Data Changes. IEEE 36th IEEE International Symposium on Software Reliability Engineering (ISSRE), Sao Paulo, Brazil, 2025. [48]*

## 6.1 Introduction

The term capacity management refers to ensuring that an IT service has sufficient infrastructure and resources to meet the current or future demand. Although capacity management is crucial to ensure efficient and effective service delivery, this process used to be carried on manually by continuously collecting and analyzing data [139]. Manual techniques to predict the capacity requirements become difficult to scale as the capacity management data sources increase, and it is significantly time-consuming for the engineers in charge.

To automate the capacity management for machine utilization, like CPU and memory, companies have started employing forecasting AIOps models, which predict the resource demand in a timely fashion. This is particularly relevant for our industry partner, ING (International Netherlands Group) Bank, where operational engineers must monitor numerous time series to ensure sufficient resources are allocated for its large-scale online operations, supported by thousands of machines with varying resource demands. As our case study, we use a capacity forecasting model developed within ING by data scientists. This forecasting model is trained on historical time series operational data related to CPU and memory utilization and predicts the number of necessary resources for the upcoming two weeks. By leveraging this model, ING significantly reduces the manual effort required to analyze historical data and predict future demand, while also minimizing the risks of insufficient or excessive resource allocation.



Real-world operational data usually has an evolving character, meaning that it constantly changes over time as it is influenced by shifts in customer behavior or infrastructure. [17, 37, 47]. For instance, a common maintenance task, such as software or hardware updates, is known to induce fundamental changes in operational data [8]. Although changes in data, commonly referred to as concept drift [15], are known to impact the performance and reliability of AIOps forecasting models over time [139, 140], their specific impact on forecasting accuracy has yet to be systematically quantified. Therefore, one of the contributions of the work is an empirical study of forecasting models' performance degradation due to data changes over a predefined period of time in a real-world industry setting. Our approach can be systematically used for other time series forecasting applications within various other industries.

One solution to overcome the issue of continuous data changes is periodic retraining [17, 18, 37, 47, 140]. Industry researchers [140] have highlighted concerns about the scalability of periodic retraining due to the computational costs of managing hundreds of time series. Therefore, our study further contributes by analyzing the benefits of continuous retraining.

Although previous work has showcased the potential of using data change (drift) detection techniques with AIOps anomaly detection models [47], to the best of our knowledge, this is the first study that investigates the benefits of retraining the capacity forecasting model within a large real-world software organization. Therefore, the third contribution of our work is investigating the benefits of retraining the capacity forecasting model used within the chosen case company only when data have changed, vs. retraining periodically. We are further presenting the implications of employing a drift detector in a practical setting, as well as retraining costs reduction, and model performance changes.

## 6.2 Motivational Use Case

In order to improve capacity management, ING is using a forecasting model which aims to predict the necessary resources for a predefined period of time. This model is currently applied to an experimental size of 16 time series (CPU and memory utilization time series) extracted from eight machines as a proof of concept, which will be used for this study. It is expected that in the future this solution will be scaled up to a significantly higher number when it is made generally available as a service in the case company.

To ensure that the forecasting model is continuously updated and aligned with potential changes in the data model, retraining is required. The model is retrained according to a retraining scheduler, and each time series has its own forecasting model. The retraining scheduler is currently programmed to retrain the forecasting model on a periodic basis every month. Although retraining more frequently, such as weekly or biweekly, could be beneficial, the responsible data scientist encountered scalability issues in terms of the computation required to retrain the model for all time series. For instance, even if the duration of retraining, including hyperparameter optimization and training the model with new data, is relatively short per time series instance when being scaled to a high number of time series, it becomes substantial. Therefore, although retraining more frequently, such as weekly or bi-weekly, could be beneficial, the models would require too much time to be updated, given the amount of data that we continuously collect. Besides this, regarding model scalability, applying our forecasting model to even more time series will further considerably increase the retraining. Furthermore, the process of redeploying a model after retraining in big organizations also implies plenty of validation. For this reason, we considered that monthly retraining is the best choice for our particular use case.



Another benefit of reducing the time spent on continuous retraining is the potential improvement in forecasting model performance. Currently, the hyperparameter tuning phase during retraining is limited to a small set of hyperparameters due to time constraints. By reducing the number of models that need retraining, the server can allocate more time to explore additional configurations. With fewer retrainings required overall, the server can free up resources that would otherwise be spent on retraining all the models for different time series. This has the potential to enhance the model's accuracy and robustness. Therefore, transitioning from periodic retraining to retraining only when necessary due to data changes can not only reduce the time needed for updates but also potentially improve model performance by allowing more hyperparameter configurations to be explored.

## 6.3 Research Questions

Although retraining only when data changes is a promising approach, we have to investigate the implications of doing so in terms of model performance and reduction in retraining time. For this reason, we begin this study with an understanding of whether we observe a severe model degradation over time when the model is never updated. This would allow us to understand the evolution of the model's performance over time. We further compare two scenarios: retraining monthly, which is our current practice, and retraining only when data changes are signaled by a drift detector. In this study, we aim to answer the following research questions:

**RQ1:** What is the evolution of the performance of the forecasting model over time?

**RQ2:** What is the difference between retraining based on drift detection and periodic retraining in terms of forecasting model accuracy and retraining frequency?

## 6.4 Related Work

This section begins with an overview of the AIOps domain and the challenges faced by AIOps models due to changes in data over time, particularly in applications dealing with time series data. We then introduce drift detection techniques tailored for time series, providing a detailed explanation of the drift detector selected for this study.

### 6.4.1 AIOps

AIOps refers to employing Artificial Intelligence (AI) and Machine Learning (ML) techniques to solve complex DevOps challenges [104]. The adoption of AIOps has the potential to decrease operational costs through automation, enhance engineering productivity, and ensure high-quality services by predicting possible failures or system anomalies [20]. A recent survey by SalesforceAI classifies AIOps applications into four categories: failure prediction, incident detection, root-cause analysis, and automated actions. Examples of failure prediction AIOps applications are predicting hard disk drives (HDDs) in large data centers [17, 18, 26, 37, 85] for hardware management purposes, node failure prediction in large-scale cloud service systems [107], and job failure prediction [17, 18, 37, 87]. In their work, Kapel et al. [141] present incident detection techniques that were successfully applied in industry, namely [142] and [143]. Root-cause analysis applications aim to reduce the time required to identify the cause of an incident by identifying abnormal patterns in Key Performance Indicators (KPIs) [27, 144]. The automated actions category aims to automate tasks performed manually by operational engineers, such as automated remediation, auto-scaling, and resource management [104]. Our studied use case, capacity forecasting, is part of the automated actions - resource management category.



### 6.4.2 Adapting AIOps Models to Data Changes

Given that changes in the operational data impact the performance of AIOps solutions, plenty of attention has been paid to adapting different AIOps models to changes in the data over time. Lyu et al. [17, 37] have shown that failure prediction AIOps models require periodic updates to preserve the accuracy over time. Moreover, in [17] the authors claim that a higher updating frequency usually leads to better performance. Anomaly detection AIOps models have also been investigated in terms of adaptation to concept drift based on retraining [47, 145]. Furthermore, besides periodic retraining, retraining AIOps models based on concept drift detection has been proposed [42, 47, 145]. However, to the best of our knowledge, no study analyzes the suitability of retraining based on drift detection when it comes to capacity forecasting models.

### 6.4.3 Drift Detection for Time Series

In their exhaustive survey about concept drift detectors, Bayram et al. [14] highlighted that, while plenty of drift detectors were developed for classification problems, few drift detectors exist for time series. The reason for this discrepancy is the lack of availability of relevant time series datasets. In this subsection, we provide a general overview of the

existing concept drift detection techniques for time series, and we explain the functionality of the drift detector we employ in this study.

#### General Overview Drift Detection for Time Series

The comprehensive study of Bayram et al. [14] reveals that while there has been considerable research on drift detectors for classification problems, there has been significantly less focus on those suitable for time series data. One reason for this is the lack of availability of open-source data on which the drift detection performance could be evaluated.

The most popular drift detector for time series is Feature Extraction Drift Detection (FEDD) [132]. This drift detector was previously applied to real-world stock market data provided by Yahoo Finance [146]. Another proposed drift detector for time series data is Entropy-Based Time Domain Feature Extraction (ETFE) [147], which was only evaluated on synthetic data. Both drift detectors require a feature extraction phase in which features are computed from each time series. However, ETFE is more computationally intensive since the features it requires are based on time series decomposition, while FEDD does not perform additional transformations of the time series. Due to scalability issues while continuously computing the features required for drift detection for each time series in real-time, ETFE is deemed unsuitable since reducing computation is a key requirement for the forecasting application. Given that efficiency is a major requirement and FEDD is less computationally intensive, we decided to employ FEDD in our experiments. Furthermore, due to internal output extraction policies, we exclude drift detectors that continuously measure the performance of the forecasting model over time to identify significant drops, such as DDM [58], EDDM [59], or ADWIN [60] and solely focus on detectors that identify drift from the time series data.



#### FEDD

FEDD is a drift detection technique designed for time series data that identifies drift based on features extracted from the time series itself. Thus, FEDD identifies drift solely from the changes that can be observed in the time series, without considering how these changes impact the performance of a forecasting model that uses the data.

The chosen drift detector consists of two main components: the feature extraction module and the drift detection module. The feature extraction module calculates time series features that will be used to detect changes. FEDD takes into account six linear features (variance, autocorrelation, skewness coefficient, partial autocorrelation, turning point rate, and kurtosis coefficient) and two non-linear features (mutual information and bicorrelation). The drift detection component computes the similarity between two feature vectors corresponding to two predefined time series windows using the cosine distance. Thereafter, the exponentially weighted moving average (EWMA) is employed to understand whether the similarity is significant and the drift needs to be signaled.

In terms of practical functionality, an initial reference time series window needs to be defined, and the initial feature vector needs to be extracted from this window. FEDD works in an online manner, namely, the reference window is constantly shifted by one sample to define the current time series window and to extract the current feature vector. The similarity between the two feature vectors, initial and current, is computed and stored in an array from which EWMA identifies whether there was a significant change in similarity. Once a drift is identified, the reference window needs to be redefined. To avoid a high

number of false alarms, the reference window is shifted by a predefined number of samples. In a practical setting, this shift implies that the drift detectors go to a cool-down period in which they will be inactive until enough samples are collected to reinitialize the reference window. Once the new reference window is defined, the drift detector restarts and is able to monitor data changes in the time series again.

## 6.5 Methodology

In this section, we present our experimental setup in terms of the datasets used, a detailed explanation of the forecasting models employed in this study, the evaluation metric used to assess the performance of the forecasting model, and the performance of retraining based on drift detection.

### 6.5.1 Datasets

We use proprietary datasets consisting of 16 time series, with half related to memory utilization and the other half to CPU utilization. These datasets were collected from real-world ING servers over a period of approximately nine months (from February until November 2023). The data was originally collected at a minute-level granularity, but to ensure confidentiality, we aggregate it to one hour by taking the mean of the samples. Although we only experimented with 16 time series, these time series are representative of data extracted from real-world financial infrastructure.



### 6.5.2 Forecasting Model Description

We employ the forecasting model that ING is currently using it to predict resource capacity in terms of CPU and memory utilization. In this subsection, we briefly present the forecasting model's functionality and the features used to train it.

#### Forecasting Functionality

The forecasting model is designed to predict CPU and memory utilization for a two-week horizon. It is applied weekly to forecast the upcoming two weeks. As a result, the forecast for the first week is expected to be the most accurate, while the prediction for the second week is more estimative but still essential for effective resource management. To simplify the evaluation, we consider that both weeks should be predicted accurately when computing the evaluation metric of the forecasting model. The forecasting model is initially trained on approximately one-third of each available time series, leaving the rest for testing and experimentation of the effect of retraining.

In order to predict the next sequence, different features are extracted from the time series. A detailed overview of the time series features is presented in the following section. To ensure that the forecasting model can predict the future using information from the present, these features are also calculated taking into account a forecasting horizon. For instance, the sample in the time series collected at the current moment (the true label) corresponds to the features computed two weeks ago.

For each time series, a LightGBM regressor is trained using the first eight weeks of data and validated in the following two weeks. The best parameters for the LightGBM model are determined using the validation set. The remaining weeks are employed to evaluate the effects of retraining the model based on drift detection vs. periodically.

#### Features

The forecasting model employs different categories of features for time series prediction, namely time, lag, rolling window, and Prophet features.

The *time features* of each time series sample refer to attributes related to the specific timestamp of the sample. These can include binary features, such as whether the sample was collected during the weekend or at the start/end of the month, as well as non-binary features like the month, quarter, or day of the month when the sample was recorded. These features are derived solely from the date of collection, without considering the actual value of the time series. The *lag features* are features computed by taking into account values of the time series in the past. These features work under the assumption that the current value of the time series is influenced by its past values, and the past values are meaningful in predicting future values. These features are computed based on a predefined forecasting horizon.

The *rolling window features* are also computed by taking into account past values of the time series. These features require a predefined time window, referring to the window required to calculate the features.

The *Prophet features* are commonly used features in machine learning applications for forecasting at scale [148]. These features are useful when time series present strong seasonal patterns or trends. Prophet features are generated by decomposing each time series into three main components, namely seasonality, holidays, and trend. In our situation, Prophet features were added to the model since the resources are used differently according to the specific time of the week, month, or year.



### 6.5.3 Retraining based on Drift Detection

In this subsection, we present how we integrated the drift detector with our forecasting model setup. We highlight some challenges that we encountered in employing FEDD in a real-world machine learning application and propose a machine learning system design that includes drift detection-based retraining.

#### Integration Challenges

While designing the architecture for the forecasting model to be retrained based on drift detection, we encountered two primary challenges: handling missing data and determining the optimal retraining timing.

**Missing Data.** In real-world scenarios, gaps in time series are inevitable due to system failures or updates that interrupt data collection. This does not affect our existing forecasting model, as we use a forecasting algorithm that is resilient to missing data. However, during our experiments, we observed that missing data presents challenges when detecting data drift using FEDD, which had not been an issue in previous evaluations on open-source continuous data. The problem arises because FEDD relies on time series features to detect drift, and these features can only be computed if the time series is continuous. To address this in our drift detection process, we treat the missing values as absent and reconstruct a continuous time series using the available data until a specified point.

**Retraining Moment.** An approach recommended in literature to overcome the impact of data changes in financial time series is presented by Cavalcante et al. [149]. The authors

propose to perform online retraining, namely learning with every new upcoming sample, once the drift is detected, until the prediction error drops. However, this approach is not feasible for our use case since we use a batch-learning approach, namely, we retrain the model once a specific size of samples (e.g., one week of new samples) is collected. For this use case, switching to online learning is not feasible. The reason for this is that our model learns from the features derived from the time series instead of the time series itself. Some time series features are calculated based on a predefined window of samples that are taken into account. An online learning approach would imply that the features have to be continuously computed and updated with every new sample, which is computationally intensive given the number of time series to which this forecasting model will be applied. Furthermore, it has been shown that online learning is impacted by the irregularity of data caused by missing values [150]. The fact that our time series contains a significant amount of missing samples is another reason why we did not include an online learning approach in our solution to handle drift.

#### Proposed Solution

To overcome the two encountered challenges, missing data and retraining moment, in this subsection, we describe our proposed solution in terms of drift detection integration.

In Fig. 6.1, we depict an example of a real-world time series from our datasets to explain how we propose to retrain the forecasting model based on drift detection. We start with an existing train and validation set that is used to train and tune the forecasting model. The model is further employed to predict the following two weeks as aforementioned. At the end of each week, we verify whether there was any drift signaled by the drift detector. If no drift is detected (e.g., the situation for Test 1), then the model is not updated. If drift is detected at the end of that week (e.g., the situation for Test 2), the model is retrained with all the available data at the end of the week where drift was identified and redeployed into production. Thus, the new model is used to obtain predictions for the following weeks.

In Fig. 6.2, we depict the workflow of the drift detector. From its original implementation, FEDD identifies drift in a real-time manner, namely that with every new sample of a time series, FEDD evaluates where drift has occurred. To integrate such functionality in a real-world case, taking into account the aforementioned limitations of this drift detection technique, at each point in time when the data should be collected, we determine whether the time series sample is missing or not. If there is a missing sample, then drift detection is not possible, and we have to wait until the next timestamp to collect data. In case of missing data, we do not perform interpolation in the time series. We solely ensure the continuity of the time series by concatenating the following non-missing sample to the current time series and therefore perform drift detection. If the sample is not missing, then we have to determine whether a drift was detected. In our setting, due to seasonality reasons, the reference window is set to eight weeks, and the current window is set to two weeks, the equivalent of a testing window. The next steps are extracting features out of the reference window and current window, respectively, storing the similarities in an array, and employing EWMA to identify whether there was a drift or not in the current testing window. Once a drift is detected, the reference window needs to be changed, and its corresponding feature vector has to be recalculated. The reference window has a fixed length, and it is changed every time a drift is identified by shifting the time window

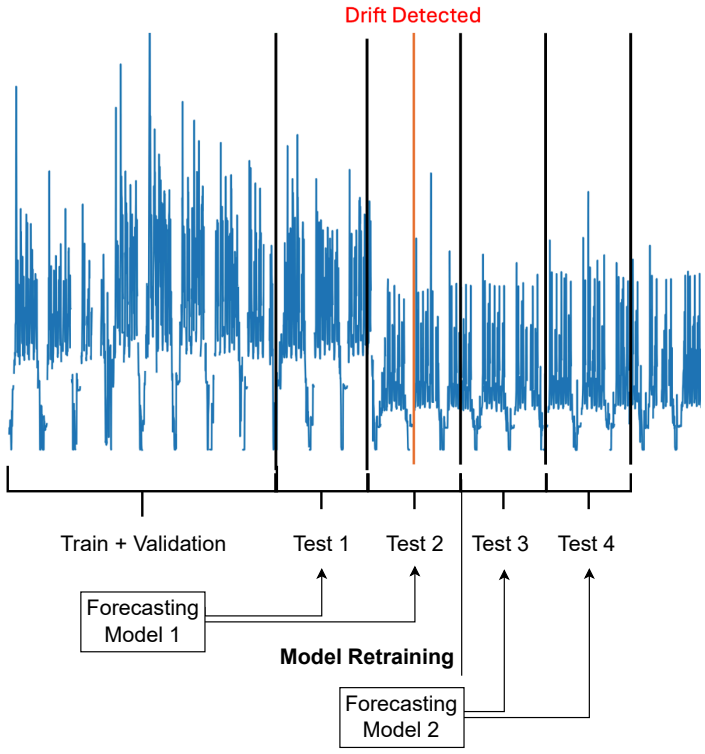



Figure 6.1: Retraining the forecasting model based on drift detection.

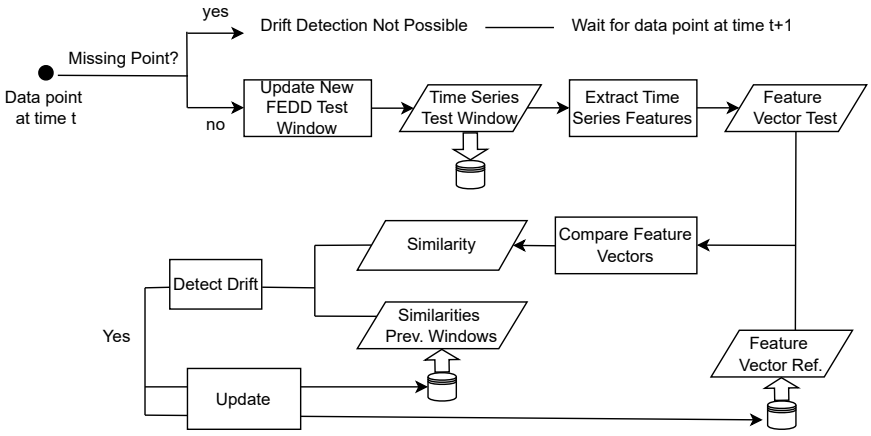



Figure 6.2: Drift Detection Block components and functionality.

considered for the reference data. By doing so, we can avoid the drift detector signaling false alarms since it still contains samples from before the change in data is detected.

One major advantage of our proposed drift detection block is the fact that not all time series require to be stored in order to detect drift, making FEDD an ideal solution when encountering scalability issues. As illustrated in Fig. 6.2, we only need to store the current evaluated time series window, the feature vector corresponding to the reference data, and the similarities array that gets continuously updated over time. Therefore, our proposed pipeline takes into account the minimization of storage space when performing drift detection and can be further employed in other time series forecasting applications where scalability is important. Furthermore, our solution is not specific to our case study, but it can be applied to any application working with time series data, such as other forecasting applications or even time series anomaly detection. Due to privacy issues, sharing the data and code is not possible. However, to encourage reproducibility, we created an open-source repository where practitioners can find a similar setting of employing FEDD on a sample time series dataset[1].

### 6.5.4 Evaluation Metrics

In this subsection, we present the evaluation metrics we employ in our study. The first part focuses on the evaluation metric used to assess the performance of the forecasting model. The second part presents how we assess the benefits of using drift detection-based retraining compared with our current monthly retraining technique.



#### Forecasting Model Evaluation Metric

To evaluate the performance of our forecasting models, we employ the mean absolute scaled error (MASE) [151]. This metric measures the error made by a forecasting model compared to a naive forecasting approach. The formula used to compute MASE is the following:

$$\text{MASE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{Y_t - \hat{Y}_t}{\frac{1}{n-1}\sum_{i=2}^{n}|Y_i - Y_{i-1}|}\right| \tag{6.1}$$

where $n$ is the number of data points in the time series, $Y_t$ is the actual value at time $t$, $\hat{Y}_t$ is the predicted value at time $t$ and $Y_{t-1}$ is the previous value of the time series

Since it is an error metric, a lower MASE is desired when creating a forecasting model. This metric is chosen to evaluate our capacity forecasting model since it is robust to outliers that can occur in capacity forecasting due to periodic fluctuations in the systems' workloads [152].

#### Evaluation Metrics for Assessing Drift Detection vs. Monthly Retraining

In this section, we are presenting how we measure and assess the difference between retraining based on drift detection and our currently implemented scenario, monthly retraining. We compare the two retraining techniques from two perspectives: the *model's performance* and the *cost reduction*.

[1]Replication Package

In terms of model performance, the first assessment metric that we calculate is the MASE improvement percentage of retraining based on drift detection (FEDD) over monthly retraining. The MASE improvement percentage can be either negative, meaning that the model retrained based on FEDD achieved a lower performance than the model that was retrained monthly, or positive, showing that the model retrained based on drift detection achieved a higher performance than the one retrained monthly. We calculate the MASE improvement percentage using the formula 6.2 depicted below. Since MASE is an error-based metric, lower MASE values correspond to a better model. A negative MASE improvement suggests that the periodically retrained model obtained better predictions than the one retrained based on FEDD. In this situation, MASE_FEDD and MASE_Periodic are calculated by averaging the MASE over all time series batches used to test the forecasting model.

$$MASE\_Impr. = \frac{MASE\_Periodic - MASE\_FEDD}{MASE\_Periodic} * 100 \tag{6.2}$$

In terms of cost reduction, we analyze the *retraining savings* while retraining based on drift detection vs. monthly retraining. We define retraining savings (RS) in terms of to which we reduce the number of times the model is retrained by employing a drift detection retraining approach vs. a periodic retraining approach. RS are calculated using the formula 6.3 shown below:

$$RS = \frac{\#Retrainings\_Periodic - \#Retrainings\_FEDD}{\#Retrainings\_Periodic} * 100 \tag{6.3}$$



## 6.6 Experimental Results

In this section, we present the results obtained during the experimentation. Each subsection corresponds to the results of the experiments designed to answer each research question. Statistical significance of performance differences between settings was assessed using the Wilcoxon signed-rank test.

### 6.6.1 Forecasting Performance over Time Analysis

With this experiment, we analyze the evolution of the forecasting model's performance over time for all 16 analyzed time series, eight corresponding to CPU utilization and eight corresponding to memory utilization. We aim to answer RQ1 by understanding whether we can observe drops in the performance of the forecasting model over time, as also analyzed in previous AIOps applications for failure prediction [17, 18, 37].

In Fig. 6.3 and Fig. 6.4, we present the performance (MASE) of the forecasting model over time for CPU and memory utilization, respectively. This MASE is depicted in the two figures by the continuous, blue line called "static". As an overview of this experiment, we cannot observe a gradual performance degradation over time for all analyzed time series. In most situations, the model's performance is relatively constant with small fluctuations. These fluctuations can indicate that retraining is needed, but the changes do not severely impact the model's performance since, in most situations, the MASE variance is smaller than 1. When it comes to CPU utilization forecasting depicted in Fig. 6.3, the time series corresponding to Machine 3 suffers a severe increase in MASE in the first weeks. However, the MASE is drastically decreasing, indicating that the remaining time series becomes

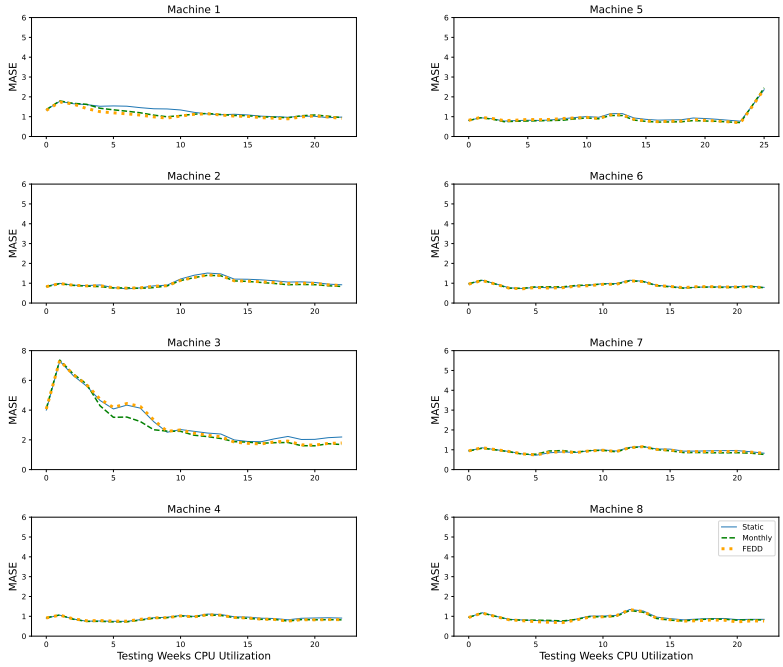



Figure 6.3: CPU Utilization

similar to the time series the model has been trained on. This suggests that the time series experiences a drastic change after a certain point and afterward changes back to its initial state. Thus, in this situation, especially, the model should be able to adapt to this temporary drift, which can be achieved by retraining the model. We can further notice in Fig. 6.3 that in the case of Machine 5, the MASE is increasing between weeks 20-25, indicating that there is a change in the data around this period.

In Fig. 6.4, we can notice that the performance of the forecasting model is gradually dropping since the MASE, which corresponds to the forecasting error, is increasing for Machines 2, 4, 6, 7, and 8. This might suggest that, for these cases, the model might benefit from being updated over time. When it comes to machines 1 and 3, it can be noticed that the drops in performance are not gradual, but rather sudden which can be observed by the fluctuations in the forecasting model's error in some specific time intervals, such as testing week 7 for Machine 3 and testing weeks 15 or 20 for Machine 1. These can be explained by fluctuations in the time series, which change suddenly and suddenly in different periods in a similar manner as in Machine 3 in the case of CPU utilization. Moreover, similar to the situation of CPU utilization, we can observe in Fig. 6.4 that Machine 5 also experiences a sudden decrease in performance (increase in MASE) between weeks 20-25.

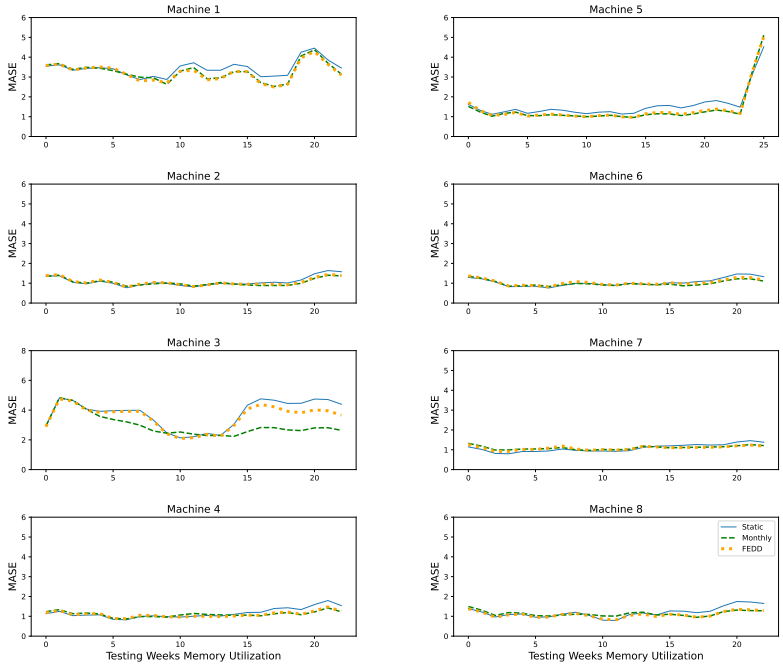



Figure 6.4: Memory Utilization

### 6.6.2 Periodic vs Drift Detection-based Retraining

In this set of experiments, we answer RQ2 and understand what is the difference between our current way of retraining the forecasting model (monthly retraining) and retraining based on drift detection using FEDD. For this experiment, the model that is retrained on a monthly basis is our baseline, since we analyze the benefits of employing concept drift detection-based retraining over periodic retraining. To understand whether FEDD-based retraining is beneficial, we measure the improvement in forecast accuracy and the percentage of retraining saved while retraining based on FEDD compared to periodic retraining. We depict our results in Table 6.1.

From Table 6.1 we can notice that in all situations, retraining based on drift detection implies 50% savings in the number of required retrainings. This suggests that the forecasting model was retrained only half of the time using the chosen drift detector (FEDD) compared to monthly. When it comes to performance improvement, we can observe that the MASE obtained by drift detection-based retraining was, for half of the time series, lower than the one obtained by retraining periodically. The most dramatic decrease in performance was observed for the time series generated by Machine 3 for both CPU and memory utilization. In these cases, the MASE decreases by 5.7% and 19.2%, respectively, when the required retrainings decrease by 50% and 67%, respectively. This shows that lowering the number

Table 6.1: MASE Improvements (Impr.) and retraining savings in percentage between retraining periodically (1 week, 2 weeks, 1 month) and retraining using FEDD. The "+" symbol shows that the model retrained based on FEDD performed better than the periodic one and the "-" shows that it performed worse.

| | Time Series | MASE Impr. (%) | Retraining Savings (%) |
|---|---|---|---|
| **CPU Utilization** | Machine 1 | +6.67 | 67 |
| | Machine 2 | -2.06 | 67 |
| | Machine 3 | -5.70 | 50 |
| | Machine 4 | +0.00 | 50 |
| | Machine 5 | -2.17 | 57 |
| | Machine 6 | +1.14 | 50 |
| | Machine 7 | -1.09 | 67 |
| | Machine 8 | +2.22 | 50 |
| **Memory Utilization** | Machine 1 | +1.52 | 50 |
| | Machine 2 | -0.95 | 50 |
| | Machine 3 | -19.20 | 67 |
| | Machine 4 | +5.36 | 50 |
| | Machine 5 | -3.73 | 57 |
| | Machine 6 | -2.97 | 50 |
| | Machine 7 | +4.42 | 67 |
| | Machine 8 | +3.54 | 50 |



of retrainings comes with the high cost of lowering the MASE of the forecasting model. In the situation of time series generated by Machines 1, 4, and 8, the MASE generated by retraining based on drift detection resulted in an increase in the forecasting performance for both CPU and memory utilization. This shows that retraining based on drift detection is beneficial for both reducing the number of times the forecasting model requires retraining and obtaining better forecasting accuracy.

Table 6.1 shows the MASE improvement when considering the average MASE on all testing batches for each retraining technique. In Fig. 6.3 and Fig. 6.4, we depict the MASE on each testing batch to better understand the MASE difference on each testing batch. The MASE corresponding to monthly retraining is depicted using a dashed line, while the MASE corresponding to retraining the forecasting model based on drift detection is depicted using a dotted line in the two figures. In both CPU and memory utilization, we do not observe significant differences in MASE between retraining monthly vs. retraining based on drift detection, except for the time series generated by Machine 3. For this particular machine, the forecasting model retrained based on drift detection performs similarly to the situation in which it was never retrained, but its MASE starts decreasing after testing batch 10 for CPU utilization (Fig. 6.3) and 15 for memory utilization 6.4. However, for this situation, the MASE obtained by retraining periodically is, in most situations, substantially lower than the one obtained by retraining based on drift detection, suggesting that for this particular time series, retraining more often is beneficial. This can also suggest that the FEDD drift detector did not manage to capture all the situations in which retraining was required.

## 6.7 Discussion

Through this case study, we noticed that employing a drift detection-based retraining approach is beneficial in the context of capacity forecasting for our industry partner. Our conclusions are derived from the fact that we demonstrated how a drift detection-based retraining reduces the time required to update the forecasting models, while minimally impacting their forecasting performance. Furthermore, employing a drift detection-based retraining approach improves the scalability of the forecasting model to multiple time series. In this section, we are further discussing the implications of retraining the capacity forecasting model using drift detection. We evaluate these implications by examining the model's accuracy over time, highlighting scenarios where drift detection-based retraining underperformed compared to monthly retraining, and explaining the potential reasons for this. Additionally, we explore the design considerations for machine learning systems when implementing drift detection-based retraining, emphasizing key factors that machine learning engineers must take into account.

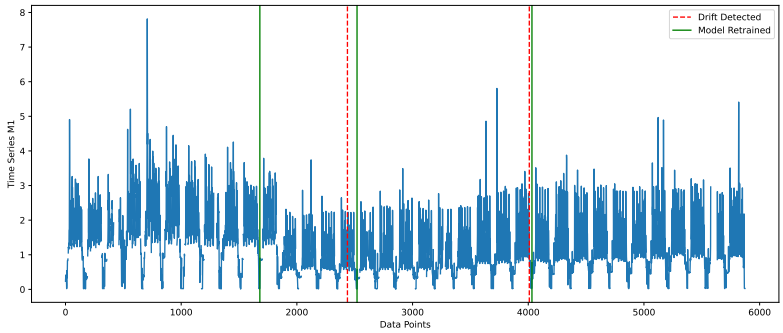
### 6.7.1 Accuracy Implications of Retraining based on Drift Detection

From our experiments, we observed that in most of the situations, the performance of the forecasting model is not highly impacted when retraining based on drift detection using FEDD compared to our current retraining practice, monthly retraining. In some situations, we observed even an improvement in forecasting accuracy when retraining based on drift detection. This shows that retraining when FEDD identifies changes in the time series data is a promising solution to reduce the number of times the forecasting model requires retraining, while not significantly impacting the model's accuracy. However, this conclusion does not hold when it comes to the time series generated by Machine 3 as observed in both Fig. 6.3 and Fig. 6.4. In this specific case, we observe that the model that is periodically retrained performs significantly better than the model retrained based on drift detection. Thus, we performed an in-depth analysis of this case, and we present our findings in this section. We are using Fig. 6.5 to explain our findings.

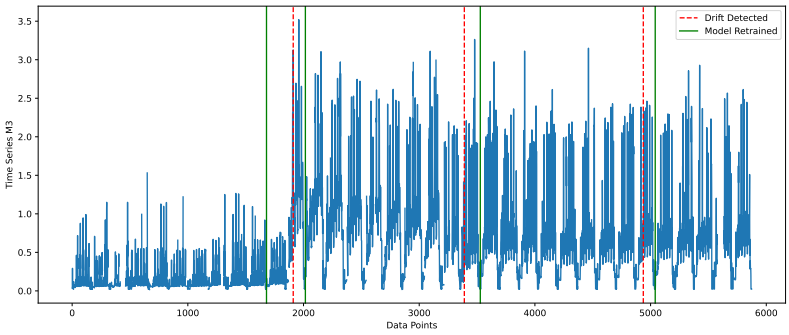
One interesting finding from our analysis is that while all selected time series generated by all machines experience changes in the data, the time series generated by Machine 3 are different in terms of how frequently they experience data changes and how long the changes last. Thus, a major characteristic of the time series generated by Machine 3 is the short and sudden changes in the data, as can also be seen in Fig. 6.5b.

We investigated whether the type of data change, a sudden change in the time series or a more gradual change in the time series, is a limitation of the FEDD drift detector. However, from Fig. 6.5a we can notice that FEDD managed to identify both a sudden data change (first identified drift) and a more gradual data change (second identified drift), in this example corresponding to Machine 1. We observed similar behavior in other time series but only decided to show Machine 1 since the difference between the data changes can be visibly noticed. Thus, the type of data change is not a limitation of FEDD, but the fact that the changes in the time series are short and might be related to the way FEDD was designed.

Once it detects a drift, FEDD enters a period of inactivity until it gathers enough data to start detecting again. During this period, according to the detector's functionality [132],




(a)




(b)

Figure 6.5: Time Series Corresponding to CPU utilization for Machine 1 (a) and Machine 3 (b) including the moments when the drift was detected by FEDD and the moments when the forecasting model was retrained.

FEDD shifts its reference data with a predefined window size to avoid continuously triggering false alarms. This period of inactivity can be the main reason why FEDD did not find any drift between data points 2000 and 3000 from Fig 6.5b, although the data changed significantly. Also, the first model retraining around data point 2000 did not capture enough changed data to allow the forecasting model to learn the new time series pattern, which can explain the low performance. This argument is further supported by the fact that the forecasting model retrained periodically achieved significantly better performance than the one trained based on drift detection. Therefore, our findings suggest that FEDD is a suitable solution as long as the changes in the time series are not frequent. For these types of time series, periodic retraining is the most suitable solution to preserve the forecast performance. Therefore, we consider building a hybrid retraining approach, in which time series with short sudden changes (e.g. the ones generated by Machine 3) are retrained periodically, while the others are retrained based on drift detection. This hybrid approach will allow one to reduce the retraining time, while ensuring that the forecast performance

is not impacted.

### 6.7.2 Forecasting System Design Implications

When it comes to designing a drift detection-based retraining pipeline for a time series forecasting model, we consider FEDD a suitable solution when scalability is a major requirement. The reason for this is that FEDD does not require storing the entire time series, and continuously accessing it to detect drift, which could come with significant storage expenses and latency when accessing the data.

A major concern that practitioners need to consider is handling missing data, as FEDD is not designed to handle missing samples. Although in our solution we opted to remove them, this might have implications for the way FEDD detects drift. For instance, a higher number of missing data points could result in a distorted time series, which FEDD could see as a drift and erroneously signal the need to retrain the model. Although in our experiments, we did not encounter this situation, we recommend practitioners be aware of this limitation of FEDD when applying it to real-world data. To avoid false alarms generated by missing samples, we suggest integrating FEDD with a missing values monitoring block to better investigate whether the drift was signaled due to a high number of missing data points. Furthermore, we recommend that researchers incorporate missing data handling techniques in time series drift detection and assess their implications in drift detection accuracy.

In terms of the moment when the forecasting model is updated, we did not consider retraining the model immediately after a drift was detected, since it is unsuitable in practical scenarios. The reason for this is the availability of a responsible data scientist to ensure that, after retraining, the model can be securely and safely deployed. For instance, if a drift is detected outside of working hours, the data scientist is not available to perform model quality checks after it has been retrained and redeployed.



## 6.8 Threats to Validity

In this case study, we focus mainly on the forecasting of capacity for CPU and memory resources using data from ING. However, this use case offers a unique and realistic setting, a large-scale financial infrastructure where time forecasting models are continuously exposed to evolving patterns in the data. Therefore, the results highlight the effectiveness of the process presented. Incorporating concept drift detection-based retraining in forecasting pipelines offers valuable insight for practitioners navigating similar contrasts in high-stakes environments.

## 6.9 Conclusions and Future Work

This work investigates the impact of retraining a time series forecasting model for capacity management based on drift detection. To identify drift and, therefore, the need to retrain the forecasting model, we employ the FEDD drift detector. In our experiments, we investigate the effect of retraining based on FEDD compared to retraining monthly in terms of the performance of the forecasting model and how many times the model requires retraining. The goal of this study is to understand whether FEDD can be used to reduce the number of times the forecasting model requires retraining, since with time, if the model is applied to

more time series, we might encounter scalability issues that will no longer allow monthly model updates.

Our experiments suggest that for both CPU and memory utilization-related time series, retraining based on drift detection does not imply a significant drop in the forecasting model's performance. The only situation in which we observed that the periodically retrained model predicts significantly better than the one retrained based on FEDD is for the time series that experience short and sudden changes. For these time series, particularly, we further suggest employing a periodic retraining approach. Furthermore, in this chapter, we discuss the implications of employing FEDD in a practical scenario and the challenges that we encountered in terms of integration with the forecasting model. For instance, we highlighted the fact that FEDD cannot handle missing values and should be carefully considered and evaluated in situations in which the number of missing samples in a time series is significantly high.

**Future Work:** Future work should focus on improving FEDD to work with distorted time series due to missing values. Furthermore, one major limitation of FEDD that we discovered is handling short, sudden changes in the time series due to the period of restart that FEDD requires after detecting a drift. For this reason, we consider that developing a drift detector with a shorter restart period, which does not signal a high number of false alarms, is required.

# 7

# Conclusion

This thesis contributes to the field of engineering machine learning (ML) systems, with a particular emphasis on techniques for monitoring and maintaining these systems throughout their lifecycle. All chapters address one specific type of monitoring for ML systems, namely concept drift detection-based data monitoring, which is examined across multiple types of ML systems, including classification, anomaly detection, and forecasting. Beyond monitoring, this thesis also investigates the maintenance of ML systems through drift detection-based retraining and compares it with the current state-of-the-art practice in terms of maintaining ML systems, periodic model retraining. The effectiveness of this approach is evaluated from two key perspectives: model accuracy and energy efficiency. Additionally, the study highlights the practical challenges and limitations associated with implementing drift-based retraining in real-world environments.



In the remainder of this chapter, we revisit each research question, summarize the steps taken to address them, and present the corresponding findings. Finally, we discuss the broader research implications of this work and outline future promising research directions.

## 7.1 Revisiting Research Questions

In this subchapter, we revisit our main research questions and sub-questions. In order to answer the research questions, we briefly explain the research process and explain our decisions during this process. We further highlight other findings derived from each chapter and provide a general answer to each research question.

**M-RQ1. What is the difference in drift detection performance between label-dependent supervised drift detectors and label-independent, unsupervised drift detectors?**

**M-RQ1** is addressed in **Chapter 2** of this thesis. In order to understand the differences in performance between supervised and unsupervised drift detectors, we employed both synthetic and real-world data, for which the moment when concept drift occurs is known beforehand. To ensure robustness in our findings, we experimented with both balanced and highly imbalanced classes for the synthetic data. Highly imbalanced classes imply that

there are significantly more samples corresponding to one particular class in the dataset compared to the number of samples from the second class. We evaluated the drift detectors' performance in terms of latency, showing how late a drift is detected after it occurs, the false positive rate, and the percentage of non-drifts signaled as drifts. We further filtered out drift detectors based on their inconsistencies when changing the random seed using the miss-detection probability metric.

Our research revealed that supervised drift detectors are more effective than unsupervised ones in identifying drift in datasets containing encoded categorical features. The reason for this is that encoding increases data sparsity, which affects data distribution estimation, a crucial step in unsupervised drift detection. In contrast, supervised drift detectors do not depend on data distribution estimation, so their performance remains unaffected in these circumstances. Supervised drift detectors also outperform unsupervised drift detectors when identifying abrupt or gradual drift with a relatively small drift width. However, supervised drift detectors are prone to a significant number of false alarms when identifying gradual drift with a larger drift width. Furthermore, neither supervised nor unsupervised drift detectors were affected by the presence of noise in the data.

In addition to addressing M-RQ1, this study presents key findings that enhance the understanding of how to apply unsupervised drift detectors in real-world scenarios. We found that data must be scaled to prevent widely distributed attribute values when using unsupervised drift detectors. Additionally, careful experimentation is necessary to determine the most suitable similarity metric for these detectors. Lastly, our research did not yield clear conclusions on the impact of class imbalance on drift detection performance. This uncertainty arose from a significant discrepancy between our findings on synthetic data and real-world data. Consequently, Chapters 2 and 3 focus specifically on an ML application that faces a severe class imbalance issue.



**Answer M-RQ1.** Supervised drift detectors are more robust when applied to sparse data and when identifying abrupt or gradual drifts with small drift width. In contrast, the performance of unsupervised drift detectors is severely impacted by data sparsity, but they outperform supervised drift detectors in identifying drifts with larger drift widths. While unsupervised detectors require data scaling before distribution computation, supervised detectors do not. Both types of drift detectors are not impacted by noise when detecting drift.

**M-RQ2. What is the impact of retraining failure prediction ML systems based on unsupervised drift detectors?**

**M-RQ2** is addressed in the following two chapters. To answer this research question, we focus on one specific AIOps application, namely failure prediction, which can be seen as an ML classification application with high class imbalance. Thus, **M-RQ2.a.** is addressed in **Chapter 3** and **M-RQ2.b.** is addressed in **Chapter 4**.

**M-RQ2.a. To what extent can retraining failure prediction ML systems based on unsupervised drift detectors mitigate their degradation in accuracy over time?**

The purpose of this study is twofold: understanding which unsupervised drift detectors are able to capture concept drift and understanding the effect of retraining failure prediction systems based on drift detection. We explore two types of unsupervised drift detectors, namely, drift detectors based on monitoring the skewness of the features over time and drift detectors based on data distribution. We experiment with two failure prediction problems, namely job and disk failure prediction.

Our study reveals that drift detectors relying on monitoring feature skewness over time should not be used as drift indicators in failure prediction. Additionally, monitoring the percentage of changed features is not an accurate method for detecting drift in data. However, our findings demonstrate that unsupervised drift detectors based on data distribution can effectively identify drift, and retraining failure prediction models based on their output achieves performance comparable to periodic retraining. Unlike the previous main research question (M-RQ1), which is inconclusive for classification problems with highly imbalanced classes, this study confirms that unsupervised drift detectors accurately detect drift in AIOps failure prediction applications. However, selecting the most suitable drift detection technique for one application requires experimentation.

**M-RQ2.b. To what extent can retraining failure prediction ML systems based on unsupervised drift detectors reduce the energy consumed by this ML application over time?**

This study aims to understand whether practitioners should consider including drift detection-based retraining when designing sustainable failure prediction systems. Besides this aspect, in this work, we explore multiple retraining techniques for failure prediction models and analyze their impact on the energy consumed during training and inference.



This chapter highlights that retraining based on drift detection can reduce the energy consumption of failure prediction applications over time, but only if the drift detector is properly selected. If the detector is overly sensitive to data changes, the energy consumed by retraining based on drift detection becomes comparable to periodic retraining. However, when the detector fails to identify drift, the accuracy of failure prediction models can significantly degrade. Therefore, designing a sustainable failure prediction system requires carefully balancing energy consumption over time with predictive accuracy when selecting the most suitable drift detector.

Other findings derived from this study show that the retraining strategy only affects the energy consumed during training, while not impacting the energy consumed during inference. Furthermore, retraining on only the most recent data is usually preferred over retraining using all available data when designing sustainable ML systems.

**Answer M-RQ2.** Retraining failure prediction models based on drift detection can be beneficial if the detector is well-suited to the dataset. A suitable detector is one that balances sensitivity: it is not too sensitive to trigger false alarms, but still able to capture the relevant changes in the data that could impact the model's accuracy. Compared to periodic retraining, drift detection-based retraining reduces the frequency of updating the model as well as the energy consumed during retraining without significantly

affecting predictive accuracy.

**M-RQ3. What is the impact of retraining AIOps applications designed for time series based on drift detection designed for time series data?**

**M-RQ3** is addressed in Chapters 5 and 6. To answer this research question, we focused on two specific AIOps applications designed for time series data, namely anomaly detection and capacity forecasting. Thus, **M-RQ3.a.** is addressed in **Chapter 5** and **M-RQ3.b.** is addressed in **Chapter 6**.

**M-RQ3.a. What is the impact on anomaly detection accuracy of retraining these ML systems based on drift detection?**

In Chapter 5, we focus on one specific AIOps application, namely anomaly detection. Therefore, we begin by implementing the most popular anomaly detection techniques for AIOps data and evaluating their performance on two operational datasets related to real-world server metrics and internet traffic metrics. We experiment with multiple retraining techniques, namely retraining on the most recent data vs. retraining on all the available data, as well as retraining periodically vs. retraining based on drift detection. To perform drift detection-based retraining, we employ the most commonly used drift detection technique for time series data, namely FEDD.



Our results suggest that retraining one good-performing anomaly detector, LSTM-AE, based on drift detection, achieves similar results to periodic retraining for the dataset containing samples related to real-world server metrics. However, when it comes to the dataset containing metrics related to internet traffic, retraining periodically is preferred in terms of accuracy.

We further discovered that not all anomaly detection models with high accuracy that we evaluated are designed for a drift detection-based retraining setting. For instance, the most popular anomaly detection model for AIOps data, SR-CNN, identifies anomalies by learning from the behavior of multiple time series. For this reason, a change detected in only one of the time series would imply retraining the anomaly detection model.

When considering that this anomaly detector is applied at scale, which means that it is used for hundreds of time series, it is highly likely that periodically a data change occurs in at least one time series. In this situation, retraining based on drift detection would likely be equivalent to retraining periodically in terms of retraining frequency. Therefore, practitioners should consider whether the employed anomaly detection technique is compatible with a drift detection-based retraining setup when designing an anomaly detection system. Another finding from this study is that anomaly detection techniques that take the time series in the time domain as input should be retrained using a sliding window approach, in which old data is discarded. On the other hand, techniques that decompose the time series into different domains, such as frequency, achieve higher anomaly detection accuracy when employing a full-history retraining technique, where retraining is performed using all the available data.

**M-RQ3.b. What are the practical and performance implications of retraining a capacity forecasting ML system based on drift detection?**

To examine the practical implications of retraining based on drift detection, we conducted a study in collaboration with the AIOps team at our industry partner, ING Netherlands. In this study, we focused on one specific application developed by the AIOps team, namely capacity forecasting, and we conducted experiments in order to understand the accuracy implications of retraining based on drift detection, as well as highlight the challenges of designing a drift detection-based retraining setup. We employed MASE as the metric that shows the accuracy of the forecasting model and analyzed the difference between retraining periodically on a monthly basis, which is the current retraining practice, compared to retraining based on drift detection. To identify drift, we employed the same concept drift detector for time series data as in Chapter 5, namely FEDD. For this work, we employed proprietary time series data related to CPU and memory utilization.

When it comes to performance, we noticed that in approximately half of the cases, the forecasting model performed better when retrained based on drift detection compared to periodically, showing that FEDD is a promising retraining solution to reduce the number of times the capacity forecasting model should be updated. On the other hand, we noticed that periodical retraining benefits situations in which the changes in the time series data are sudden and short. This is due to the design of the chosen drift detector, since after identifying a drift, FEDD requires a period of inactivity to gather new data and restart. Thus, during this period, data can experience more changes that are not captured, and the forecasting model is not updated. In addition to periods of inactivity, we observed another challenge when using FEDD for drift detection: handling missing data. Since FEDD was neither designed nor evaluated for time series with missing values, this poses a limitation-particularly given that missing data is common in real-world scenarios. For example, the occurrence of incidents in the software system or the inactivity of the equipment collecting data can lead to incomplete time series. Therefore, our study proposes a solution for implementing a drift detection-based retraining setup for forecasting models in real-world applications, taking into consideration the limitation of missing data.



**Answer M-RQ3.** Periodic retraining generally achieves higher accuracy than drift detection-based retraining (using the FEDD drift detector) in time series AIOps applications. This might suggests that FEDD fails to capture all drifts, possibly due to its inactivity period after detecting a drift, leading to slightly lower predictive performance.

## 7.2 Research Implications and Future Work

This subsection aims to highlight the main research implications derived from this thesis, as well as promising future research directions that could facilitate the understanding of concept drift detection-based retraining in industrial settings.

**More open source real-world datasets with highlighted drifts to evaluate the accuracy of drift detectors.** One limitation we faced while conducting this research is the inaccessibility of datasets in which the moment of drift occurrence is known or labeled

by experts. Without highlighting the moment when concept drift occurs, an extensive study of concept drift detection performance, such as studying the delay in drift detection or precisely identifying the number of false alarms generated, could not be performed. In Chapter 2 we were able to evaluate to evaluate the accuracy of drift detection since we employed synthetic data, where we could set the moment of drift occurrence during data generation, and two real-world datasets, namely the ELECT2 dataset [70] and the Airlines dataset [68], where the moments when drift occurs were highlighted by previous work [69]. However, we observed that the conclusions drawn from experiments with synthetic data are in some situations different from the ones drawn from experimenting with real-world data, which could be caused by the fact that synthetic data do not always imitate the behavior of real-world data. This highlights the importance of evaluating drift detectors using real-world data to draw conclusions that can be applicable to other real-world scenarios. An attempt to overcome this limitation is presented in Chapter 3, where we linked the moment when concept drift occurs with the moment when there is a significant drop in the model's performance. Although this can be a promising solution to evaluate drift detectors when the exact moment of drift is unknown, the research community needs more datasets with documented drift occurrences to evaluate the performance of drift detectors and highlight their strengths and limitations. This needs to be a collaborative effort for both academia and industry, since the moments when drift occurs can solely be indicated by domain experts who know and understand the behavior of data.

**Future work:** In this direction, future research should prioritize the development of a standardized benchmark of real-world datasets that will be used to evaluate drift detectors. This benchmark could also be used to understand whether there are general patterns in the data behavior that are related to concept drift and to what extent these patterns can be used further be leveraged to facilitate easier drift labeling. Furthermore, analysis on bias when labeling drift on these datasets should be performed.



**Defining concept drift in different domains that employ machine learning techniques.** In addition to increasing the availability of datasets where the moment when drift occurs is known, more explanations in terms of what kind of data changes need to be identified would also improve the explainability of machine learning systems. For instance, Webb et al. [69] defined a drift in the ELECT2 dataset as the moment when new rules for the electricity market took place. Furthermore, defining what concept drift is by domain experts would be significantly relevant also for time series data, which was targeted in Chapters 5 and 6, since it would help differentiate between significant changes in the time series trend and a continuous sequence of anomalies.

**Future work:** Future research should focus on the definition and standardization of concept drift for multiple datasets and in different domains. Subsequently, it would be valuable to investigate to what extent are drift detectors mistakenly interpreting sequences of anomalies as concept drift.

**Unsupervised drift detectors have high potential, but their implementation is not always publicly available.** Our research demonstrated that unsupervised drift detectors could be beneficial to include in the retraining pipelines of ML systems, especially in the case of AIOps systems. Therefore, we advise practitioners to experiment with multiple drift detectors and select the one that is the most suitable for their application. However, although plenty of unsupervised drift detectors were proposed in the literature [19], another

important limitation of our study is the availability of open-source code for these detectors. In our work, we opted to replicate the functionalities described for these drift detectors or to rely on publicly available open-source implementations of drift detectors. However, we strongly encourage researchers developing concept drift detectors to publicly share their code to enhance accessibility for practitioners and other researchers.

**Future work:** The development of a standardized platform or toolkit where all practitioners could share their drift detection implementation would be highly beneficial for the research community. Researchers could further explore ways to standardize the platform, including unified input/output formats and comprehensive tutorials for integrating drift detectors into experimental and production-level machine learning setups. Ultimately, this kind of shared infrastructure could significantly accelerate the adoption of drift detection methods in industry.

**In line with the No Free Lunch Theorem, there is no best concept drift detector for all applications and datasets.** The no-free-lunch theorem in machine learning refers to the fact that there is no specific learning algorithm that is the best to use in every situation [153]. The performance of each algorithm depends on the dataset to which it is applied and the machine learning problem that is being tackled. Our findings suggest that this theorem also applies to drift detection. Therefore, we demonstrate that the best drift detection technique depends on the dataset to which it is applied, and, thereby, practitioners should consider experimenting with multiple drift detectors to understand the most suitable one for their problem. For this reason, future work can explore the effects of multiple drift detectors in different real-world domains.

**Future Work:** Although Chapter 6 of this thesis examined the impact of drift detection-based retraining in a real-world application, namely capacity forecasting, we believe that more research on concept drift detection in practical scenarios is needed. This would enhance our understanding of how employing concept drift detectors to monitor data changes affects the performance of ML systems over time. Moreover, expanding this research to different research domains would allow researchers to discover patterns in terms of how well drift is detected with respect to the data on which the detector is applied.



**Long-term analysis of energy consumption of ML systems in production.** In Chapter 4, we analyzed the impact on energy consumption of retraining 3 failure prediction models based on drift detection. These models are built using real-world operational data and were evaluated under varying retraining frequencies, namely daily, weekly and monthly. The retraining frequency was based on the data availability. Our analysis focused specifically on the energy consumed during three core ML lifecycle stages: training, inference, and drift detection. Therefore, it did not account for the ML stages required before model training, such as data collection and preprocessing, which can also increase the overall energy footprint of an ML system. Although in this thesis the energy consumed while retraining AIOps applications designed for time series data was not assessed, we expect these types of systems to be even more intense in the energy consumed, especially if they operate real-time. The reason for this assumption is that these systems continuously collect large volumes of data which needs to be constantly verified against drift. Furthermore, the currently evaluated drift detector for time series, FEDD, performs real-time drift checks, whereas the detectors used in failure prediction only assess drift after accumulating a certain amount of data. Consequently, we anticipate that AIOps systems

for time series will consume significantly more energy during drift detection compared to failure prediction scenarios, given the substantially higher frequency of drift checks. As for energy consumption during retraining, it largely depends on the volume of data processed and the anomaly detection or forecasting algorithm employed.

**Future Work:** Future work should focus on building on our research by including an empirical evaluation of the energy consumed by multiple stages of an ML system, such as data acquisition and preprocessing. In this way, we could have a better understanding of the total energy consumed by an ML system. Furthermore, we believe that an interesting research direction will be expanding the research performed for AIOps systems designed for multivariate data to AIOps systems designed for time series data. This could provide insights into how energy consumption during drift detection varies across different machine learning applications. Another promising direction would be to adjust the AIOps solutions explored in this thesis, namely failure prediction, anomaly detection, and capacity forecasting, according to the green AI design patterns depicted in [29] and investigating the energy consumption reduction. In this way, the most impactful green tactics would be explored and highlighted as best practices when designing green machine learning systems.

**Boosting the trustworthiness of machine learning systems through concept drift detection in practical settings.** Based on our experiments, concept drift detection is a valuable tool for monitoring data changes over time as well as helping in designing more energy-efficient ML solutions. When appropriately selected, drift detectors can effectively identify meaningful data shifts while minimizing false alarms. However, whether it has the capability to improve the long-term reliability of machine learning systems within organizations requires more exploration and experimentation.



**Future Work:** Therefore, future work should try to determine data scientists' perception of integrating concept drift detectors in their machine learning workflows. This thesis was inspired by my own experience as a data scientist, where I observed how unexpected changes in data can undermine the trustworthiness of ML systems. Building on this, a valuable next step would be to investigate whether concept drift detection can serve as an initial step in designing more reliable and robust ML systems. Furthermore, another valuable research path would be to determining whether concept drift detection enhances the trust and confidence of other stakeholders in the ML system, fostering better alignment between stakeholders and the data science. If concept drift detection proves to be beneficial in a practical setting, then continuous monitoring of ML systems should be established as a best practice in organizational ML design.

# Bibliography

## References

[1] Iqbal Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2, 03 2021.

[2] Laila Al-Blooshi and Haitham Nobanee. Applications of artificial intelligence in financial management decisions: A mini-review. *SSRN Electronic Journal*, 02 2020.

[3] Kirtan Jha, Aalap Doshi, Poojan Patel, and Manan Shah. A comprehensive review on automation in agriculture using artificial intelligence. *Artificial Intelligence in Agriculture*, 2:1–12, 2019.

[4] Jay Lee, Hossein Davari, Jaskaran Singh, and Vibhor Pandhare. Industrial artificial intelligence for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 18:20–23, 2018.

[5] David B. Olawade, Aanuoluwapo C. David-Olawade, Ojima Z. Wada, Akinsola J. Asaolu, Temitope Adereni, and Jonathan Ling. Artificial intelligence in healthcare delivery: Prospects and pitfalls. *Journal of Medicine, Surgery, and Public Health*, 3:100108, 2024.

[6] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 39–48, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344, 2017.

[8] Yangguang Li, Zhen Ming (Jack) Jiang, Heng Li, Ahmed E. Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. Predicting node failures in an ultra-large-scale cloud computing platform: An AIOps solution. *ACM Trans. Softw. Eng. Methodol.*, 29(2), April 2020.

[9] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019.

[10] Daniel Vela, Andrew Sharp, Richard Zhang, Trang Nguyen, An Hoang, and Oleg S. Pianykh. Temporal quality degradation in ai models. *Scientific Reports*, 12(1):11654, July 2022.

[11] Md Shahi Amran Hossain, Abu Shad Ahammed, Divya Prakash Biswas, and Roman Obermaisser. Impact analysis of data drift towards the development of safety-critical automotive system. In *2024 International Symposium ELMAR*, pages 325–330, 2024.

[12] Nastaran Enshaei and Farnoosh Naderkhani. The role of data quality for reliable ai performance in medical applications. *IEEE Reliability Magazine*, 1(3):24–28, 2024.

[13] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. *An Overview of Concept Drift Applications*, pages 91–114. Springer International Publishing, Cham, 2016.

[14] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 2022.

[15] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), 2014.

[16] Andreas Vogelsang and Markus Borg. Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 245–251, 2019.

[17] Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. An empirical study of the impact of data splitting decisions on the performance of AIOps solutions. *ACM Trans. Softw. Eng. Methodol.*, 30(4), July 2021.

[18] Yingzhe Lyu, Heng Li, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. On the model update strategies for supervised learning in AIOps solutions. *ACM Trans. Softw. Eng. Methodol.*, 33(7), August 2024.

[19] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

[20] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: Real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5, 2019.

[21] Anas Dakkak, Jan Bosch, and Helena Olsson. Towards AIOps enabled services in continuously evolving software-intensive embedded systems. *Journal of Software: Evolution and Process*, page e2592, 06 2023.

[22] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven C H Hoi. AI for IT operations (AIOps) on cloud platforms: Reviews, opportunities and challenges. *ArXiv*, April 2023.

[23] Tao Huang, Pengfei Chen, and Ruipeng Li. A semi-supervised vae based active anomaly detection framework in multivariate time series for online systems. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 1797–1806, New York, NY, USA, 2022. Association for Computing Machinery.

[24] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 1893–1901, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 480–490, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 39–48, New York, NY, USA, 2016. Association for Computing Machinery.

[27] Hiep Nguyen, Yongmin Tan, and Xiaohui Gu. Pal: Propagation-aware anomaly localization for cloud hosted distributed applications. In *Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, SLAML '11, New York, NY, USA, 2011. Association for Computing Machinery.

[28] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, November 2020.

[29] Heli Järvenpää, Patricia Lago, Justus Bogner, Grace Lewis, Henry Muccini, and Ipek Ozkaya. A synthesis of green architectural tactics for ml-enabled systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, ICSE-SEIS'24, page 130–141, New York, NY, USA, 2024. Association for Computing Machinery.

[30] Sasha Luccioni, Yacine Jernite, and Emma Strubell. Power hungry processing: Watts driving the cost of ai deployment? In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '24, page 85–99, New York, NY, USA, 2024. Association for Computing Machinery.

[31] Roberto Verdecchia, Luís Cruz, June Sallou, Michelle Lin, James Wickenden, and Estelle Hotellier. Data-Centric Green AI: An Exploratory Empirical Study. In *ICT4S 2022 - 8th International Conference on ICT for Sustainability*, 2022.

[32] Roberto Souto Maior Barros and Silas Garrido T. Carvalho Santos. A large-scale comparison of concept drift detectors. *Information Sciences*, 451-452:348–370, 2018.

[33] Elif Selen Babüroğlu, Alptekin Durmuşoğlu, and Türkay Dereli. Novel hybrid pair recommendations based on a large-scale comparative study of concept drift detection. *Expert Systems with Applications*, 163:113786, 2021.

[34] Paulo M. Gonçalves, Silas G.T. de Carvalho Santos, Roberto S.M. Barros, and Davi C.L. Vieira. A comparative study on concept drift detectors. *Expert Systems with Applications*, 41(18):8144–8156, 2014.

[35] Yangguang Li, Zhen Ming (Jack) Jiang, Heng Li, Ahmed E. Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. Predicting node failures in an ultra-large-scale cloud computing platform: An AIOps solution. *ACM Trans. Softw. Eng. Methodol.*, 29(2), April 2020.

[36] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 480–490, New York, NY, USA, 2018. Association for Computing Machinery.

[37] Yingzhe Lyu, Gopi Krishnan Rajbahadur, Dayi Lin, Boyuan Chen, and Zhen Ming (Jack) Jiang. Towards a consistent interpretation of AIOps models. *ACM Trans. Softw. Eng. Methodol.*, 31(1), November 2021.

[38] Xiaoyun Li, Guangba Yu, Pengfei Chen, Hongyang Chen, and Zhekang Chen. Going through the life cycle of faults in clouds: Guidelines on fault handling. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 121–132, 2022.

[39] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1123–1132, 2017.

[40] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1723–1726, New York, NY, USA, 2017. Association for Computing Machinery.

[41] Lorena Poenaru-Olaru, Luis Cruz, Arie van Deursen, and Jan S. Rellermeyer. Are Concept Drift Detectors Reliable Alarming Systems? - A Comparative Study . In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3364–3373, Los Alamitos, CA, USA, December 2022. IEEE Computer Society.

[42] Lorena Poenaru-Olaru, Luis Cruz, Jan S. Rellermeyer, and Arie van Deursen. Maintaining and monitoring AIOps models against concept drift. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, pages 98–99, 2023.

[43] Lorena Poenaru-Olaru, Luis Cruz, Jan S. Rellermeyer, and Arie van Deursen. Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring . In *2025 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*, pages 1–8, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[44] L. Poenaru-Olaru, J. Sallou, L. Cruz, J. S. Rellermeyer, and A. van Deursen. Retrain ai systems responsibly! use sustainable concept drift adaptation techniques. In *2023 IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS)*, pages 17–18, 2023.

[45] Lorena Poenaru-Olaru, June Sallou, Luis Cruz, Jan S. Rellermeyer, and Arie van Deursen. Sustainable machine learning retraining: Optimizing energy efficiency without compromising accuracy. In *2025 11th International Conference on ICT for Sustainability (ICT4S)*, pages 100–111, 2025.

[46] Miroslaw Staron, Silvia Abrahão, Grace Lewis, Henry Muccini, and Chetan Honnenahalli. Bringing software engineering discipline to the development of ai-enabled systems. *IEEE Software*, 41(5):79–82, 2024.

[47] Lorena Poenaru-Olaru, Natalia Karpova, Luis Cruz, Jan S. Rellermeyer, and Arie van Deursen. Is your anomaly detector ready for change? adapting AIOps solutions to the real world. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, CAIN '24, page 222–233, New York, NY, USA, 2024. Association for Computing Machinery.

[48] Lorena Poenaru-Olaru, Wouter van 't Hof, Adrian Stańdo, Arkadiusz P. Trawiński, Eileen Kapel, Jan S. Rellermeyer, Luis Cruz, and Arie van Deursen. Prepared for the unknown: Adapting AIOps capacity forecasting models to data changes. In *The 36th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2025.

[49] Liangwei Zhang, Jing Lin, and Ramin Karim. Sliding window-based fault detection from high-dimensional data streams. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(2):289–303, 2017.

[50] Saranya Kunasekaran and Chellammal Suriyanarayanan. Anomaly detection techniques for streaming data–an overview. *Malaya Journal of Matematik*, S:703–710, 01 2020.

[51] Suja A. Alex, Uttam Ghosh, and Nazeeruddin Mohammad. Weather prediction from imbalanced data stream using 1d-convolutional neural network. In *2022 10th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-22)*, pages 1–6, 2022.

[52] Jean Paul Barddal, Lucas Loezer, Fabrício Enembreck, and Riccardo Lanzuolo. Lessons learned from data stream classification applied to credit scoring. *Expert Systems with Applications*, 162:113899, 2020.

[53] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multidimensional data streams. *Interfaces*, 2006.

[54] Abdulhakim Qahtan, Basma Alharbi, suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams. In *KDD*, 2015.

[55] Yoav Freund and Robert E. Schapire. A short introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, 1999.

[56] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, NY, USA, 2016. Association for Computing Machinery.

[57] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[58] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[59] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method, 01 2006.

[60] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, pages 443–448, 2007.

[61] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2015.

[62] Anjin Liu, Guangquan Zhang, and Jie Lu. Fuzzy time windowing for gradual concept drift adaptation. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, page 1–6. IEEE Press, 2017.

[63] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198, 2012.

[64] Roberto S.M. Barros, Danilo R.L. Cabral, Paulo M. Gonçalves, and Silas G.T.C. Santos. Rddm: Reactive drift detection method. *Expert Systems with Applications*, 90:344–355, 2017.

[65] Feng Gu, Guangquan Zhang, Jie Lu, and Chin-Teng Lin. Concept drift detection based on equal density estimation. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 24–30, 2016.

[66] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, page 180–191. VLDB Endowment, 2004.

[67] Junming Shao, Zahra Ahmadi, and Stefan Kramer. Prototype-based learning on concept-drifting data streams. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 412–421, New York, NY, USA, 2014. Association for Computing Machinery.

[68] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, 2010.

[69] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. Analyzing concept drift and shift from sample data. *Data Min. Knowl. Discov.*, 32(5):1179–1199, September 2018.

[70] M. Harries, University of New South Wales. School of Computer Science, and Engineering. *Splice-2 Comparative Evaluation: Electricity Pricing*. PANDORA electronic collection. University of New South Wales, School of Computer Science and Engineering, 1999.

[71] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 377–382, New York, NY, USA, 2001. Association for Computing Machinery.

[72] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[73] Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. Adaptive xgboost for evolving data streams. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[74] Fang Chu and Carlo Zaniolo. Fast and light boosting for adaptive mining of data streams. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 282–292, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[75] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Model. Meth. Appl. Sci.*, 1, 2007.

[76] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[77] Elmar Plischke and Emanuele Borgonovo. Fighting the curse of sparsity: Probabilistic sensitivity measures from cumulative distribution functions. *Risk Analysis*, 40, 07 2020.

[78] Md Manjurul Ahsan, M. Mahmud, Pritom Saha, Kishor Datta Gupta, and Zahed Siddique. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9:52, 07 2021.

[79] Mark Haakman, Luís Cruz, Hennie Huijgens, and Arie van Deursen. Ai lifecycle models need to be revised. *Empirical Software Engineering*, 26(5):1–29, 2021.

[80] Anders Arpteg, Bjorn Brinne, Luka Crnkovic-Friis, and Jan Bosch. Software engineering challenges of deep learning. In *SEAA 2018*, pages 50–59, 08 2018.

[81] Abdul Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering*, 25:1–38, 11 2020.

[82] Meenu Mary John, Helena Holmström Olsson, Jan Bosch, and Daniel Gillblad. Exploring trade-offs in mlops adoption. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 369–375, 2023.

[83] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, and Dan Dennison. Hidden technical debt in machine learning systems. *NIPS*, pages 2494–2502, 01 2015.

[84] Backblaze Inc. Backblaze hard drive stats. https://www.backblaze.com/cloud-storage/resources/hard-drive-test-data.

[85] Farzaneh Mahdisoltani, Ioan A. Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *USENIX Annual Technical Conference*, 2017.

[86] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., November 2011. Revised 2012.03.20. Posted at http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[87] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344, 2017.

[88] Alibaba Group. 2021. alibaba cluster trace program. https://github.com/alibaba/clusterdata.

[89] Weng Qizhen, Xiao Wencong, Yu Yinghao, Wang Wei, Wang Cheng, He Jian, Li Yong, Zhang Liping, Lin Wei, and Ding Yu. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters. In *USENIX 2022*, 2022.

[90] Divish Rengasamy, Benjamin C. Rothwell, and Grazziela P. Figueredo. Towards a more reliable interpretation of machine learning outputs for safety-critical systems using feature importance fusion. *Applied Sciences*, 11(24), 2021.

[91] Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo A. Oliva, Yasutaka Kamei, and Ahmed E. Hassan. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*, 48(7):2245–2261, 2022.

[92] Grace A. Lewis, Sebastián Echeverría, Lena Pons, and Jeffrey Chrabaszcz. Augur: A step towards realistic drift detection in production ml systems. In *2022 IEEE/ACM 1st International Workshop on Software Engineering for Responsible Artificial Intelligence (SE4RAI)*, pages 37–44, 2022.

[93] Xiao Li, Yu Wang, Sumanta Basu, Karl Kumbier, and Bin Yu. *A debiased MDI feature importance measure for random forests*, chapter 723. Curran Associates Inc., Red Hook, NY, USA, 2019.

[94] Divish Rengasamy, Jimiama M. Mase, Aayush Kumar, Benjamin Rothwell, Mercedes Torres Torres, Morgan R. Alexander, David A. Winkler, and Grazziela P. Figueredo. Feature importance in machine learning models: A fuzzy information fusion approach. *Neurocomputing*, 511:163–174, 2022.

[95] Marco Sandri and Paola Zuccolotto. A bias correction algorithm for the gini variable importance measure in classification trees. *Journal of Computational and Graphical Statistics*, 17:1–18, 09 2008.

[96] Zhengze Zhou and Giles Hooker. Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2), January 2021.

[97] Peter J. Neumann, Theodore G. Ganiats, Louise B. Russell, Gillian D. Sanders, and Joanna E. Siegel. *Cost-Effectiveness in Health and Medicine.* Oxford University Press, 12 2016.

[98] Meenu Mary John, Helena Holmström Olsson, Jan Bosch, and Daniel Gillblad. Exploring trade-offs in mlops adoption. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 369–375, 2023.

[99] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguía, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 04 2021.

[100] Carole-Jean Wu et al. Sustainable ai: Environmental implications, challenges and opportunities. In *Proceedings of Machine Learning and Systems*, volume 4, pages 795–813, 2022.

[101] Tomaso Trinci, Simone Magistri, Roberto Verdecchia, and Andrew D. Bagdanov. How green is continual learning, really? analyzing the energy consumption in continual training of vision foundation models, 2024.

[102] Rafiullah Omar, Justus Bogner, Joran Leest, Vincenzo Stoico, Patricia Lago, and Henry Muccini. How to sustainably monitor ml-enabled systems? accuracy and energy efficiency tradeoffs in concept drift detection. In *2024 International Conference on ICT for Sustainability (ICT4S)*, 2024.

[103] Adnan Masood and Adnan Hashmi. *AIOps: Predictive Analytics & Machine Learning in Operations*, pages 359–382. Apress, Berkeley, CA, 2019.

[104] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenjing Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven C. H. Hoi. Ai for it operations (AIOps) on cloud platforms: Reviews, opportunities and challenges. *ArXiv*, 2023.

[105] Maayan Harel, Koby Crammer, Ran El-Yaniv, and Shie Mannor. Concept drift detection through resampling. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, 2014.

[106] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–24, 10 2018.

[107] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, Murali Chintalapati, and Dongmei Zhang. Improving service availability of cloud systems by predicting disk error. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 481–493, USA, 2018. USENIX Association.

[108] Yinlena Xu, Silverio Martínez-Fernández, Matias Martinez, and Xavier Franch. Energy efficiency of training neural network architectures: An empirical study. In *In proceedings of Hawaii International Conference on System Sciences*, 01 2023.

[109] Luís Cruz. Green software engineering done right: a scientific guide to set up energy efficiency experiments. *Blog post*, 2021.

[110] Benoit Courty, Victor Schmidt, Sasha Luccioni, Goyal-Kamal, Marion Coutarel, Boris Feld, Jérémy Lecourt, Liam Connell, Amine Saboni, Inimaz, supatomic, Mathilde Léval, Luis Blanche, Alexis Cruveiller, ouminasara, Franklin Zhao, Aditya Joshi, Alexis Bogroff, Hugues de Lavoreille, Niko Laskaris, Edoardo Abati, Douglas Blank, Ziyao Wang, Armin Catovic, Marc Alencon, Michał Stęchły, Christian Bauer, Lucas Otávio N. de Araújo, JPW, and Minerva Books. mlco2/codecarbon: v2.4.1, May 2024.

[111] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. A systematic mapping study in AIOps. In *Service-Oriented Computing – ICSOC 2020 Workshops*, pages 110–123, Cham, 2021. Springer International Publishing.

[112] Heng Li, Tse-Hsun (Peter) Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Adopting autonomic computing capabilities in existing large-scale systems: an industrial experience report. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, page 1–10, New York, NY, USA, 2018. Association for Computing Machinery.

[113] Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 413–425, New York, NY, USA, 2022. Association for Computing Machinery.

[114] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kastner. A meta-summary of challenges in building products with ml components – collecting experiences from 4758+ practitioners. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, pages 171–183, Los Alamitos, CA, USA, 2023. IEEE Computer Society.

[115] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference*, WWW '19, page 2659–2665, New York, NY, USA, 2019. Association for Computing Machinery.

[116] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference*, WWW '19, page 2659–2665, 2019.

[117] Ali Imran Jehangiri, Ramin Yahyapour, Philipp Wieder, Edwin Yaqub, and Kuan Lu. Diagnosing cloud performance anomalies using large time series dataset analysis. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 930–933, 2014.

[118] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. Identifying recurrent and unknown performance issues. In *2014 IEEE International Conference on Data Mining*, pages 320–329, 2014.

[119] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, 2016.

[120] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 92–103, 2020.

[121] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 3009–3017, New York, NY, USA, 2019. Association for Computing Machinery.

[122] Run-Qing Chen, Guang-Hui Shi, Wan-Lei Zhao, and Chang-Hui Liang. A joint model for it operation series prediction and anomaly detection. *Neurocomputing*, 448:130–139, 2021.

[123] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 187–196, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[124] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, May 2022.

[125] Faraz Rasheed, Peter Peng, Reda Alhajj, and Jon Rokne. Fourier transform based spatial outlier mining. In Emilio Corchado and Hujun Yin, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2009*, pages 317–324, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[126] Yu Yufeng, Yuelong Zhu, Shijin Li, and Dingsheng Wan. Time series outlier detection based on sliding window prediction. *Mathematical Problems in Engineering*, 2014, 10 2014.

[127] Oleksandr I. Provotar, Yaroslav M. Linder, and Maksym M. Veres. Unsupervised anomaly detection in time series using lstm-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pages 513–517, 2019.

[128] Li Wei, Nitin Kumar, Venkata Lolla, Eamonn J. Keogh, Stefano Lonardi, and Chotirat Ratanamahatana. Assumption-free anomaly detection in time series. In *Proceedings of the 17th International Conference on Scientific and Statistical Database Management*, SSDBM'2005, page 237–240, Berkeley, USA, 2005. Lawrence Berkeley Laboratory.

[129] Kamran Shaukat, Talha Mahboob Alam, Suhuai Luo, Shakir Shabbir, Ibrahim A. Hameed, Jiaming Li, Syed Konain Abbas, and Umair Javed. A review of time-series anomaly detection techniques: A step to future perspectives. In Kohei Arai, editor, *Advances in Information and Communication*, pages 865–877, Cham, 2021. Springer International Publishing.

[130] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 187–196, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[131] Andrea Rosà, Lydia Y. Chen, and Walter Binder. Catching failures of failures at big-data clusters: A two-level neural network approach. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 231–236, 2015.

[132] Gustavo H. F. M. Oliveira, Rodolfo C. Cavalcante, George G. Cabral, Leandro L. Minku, and Adriano L. I. Oliveira. Time series forecasting in the presence of concept drift:

A pso-based approach. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 239–246, 2017.

[133] Fengqian Ding and Chao Luo. The entropy-based time domain feature extraction for online concept drift detection. *Entropy*, 21(12), 2019.

[134] Renjie Wu and Eamonn J. Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering*, 35(3):2421–2429, 2023.

[135] Nidhi Singh and Craig Olinsky. Demystifying numenta anomaly benchmark. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1570–1577, 2017.

[136] Tao Ban, Ndichu Samuel, Takeshi Takahashi, and Daisuke Inoue. Combat security alert fatigue with ai-assisted techniques. In *Cyber Security Experimentation and Test Workshop*, CSET '21, page 9–16, New York, NY, USA, 2021. Association for Computing Machinery.

[137] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, volume 34, pages 1–5, 2017.

[138] Gustavo H. F. M. Oliveira, Rodolfo C. Cavalcante, George G. Cabral, Leandro L. Minku, and Adriano L. I. Oliveira. Time series forecasting in the presence of concept drift: A pso-based approach. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 239–246, 2017.

[139] Kun Wang, Yuan Tan, Lizhong Zhang, Zhigang Chen, and Jinghong Lei. A network traffic prediction method for AIOps based on tda and attention gru. *Applied Sciences*, 2022.

[140] Ebenezer R. H. P. Isaac and Bulbul Singh. Qbsd: Quartile-based seasonality decomposition for cost-effective ran kpi forecasting. In *2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)*, pages 847–851, 2025.

[141] Eileen Kapel, Luis Cruz, Diomidis Spinellis, and Arie Van Deursen. On the difficulty of identifying incident-inducing changes. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '24, page 36–46, New York, NY, USA, 2024. Association for Computing Machinery.

[142] Raghav Batta, Larisa Shwartz, Michael Nidd, Amar Prakash Azad, and Harshit Kumar. A system for proactive risk assessment of application changes in cloud operations. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 112–123, 2021.

[143] Sinem Güven, Karin Murthy, Larisa Shwartz, and Amit Paradkar. Towards establishing causality between change and incident. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 937–942, 2016.

[144] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*, WWW '19, page 3215–3222, New York, NY, USA, 2019. Association for Computing Machinery.

[145] Ebenezer R. H. P. Isaac and Akshat Sharma. Adaptive thresholding heuristic for kpi anomaly detection, 2023.

[146] Gustavo H. F. M. Oliveira, Rodolfo C. Cavalcante, George G. Cabral, Leandro L. Minku, and Adriano L. I. Oliveira. Time series forecasting in the presence of concept drift: A pso-based approach. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 239–246, 2017.

[147] Fengqian Ding and Chao Luo. The entropy-based time domain feature extraction for online concept drift detection. *Entropy*, 21(12), 2019.

[148] Sean Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72, 09 2017.

[149] Rodolfo C. Cavalcante and Adriano L. I. Oliveira. An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[150] Steven Cheng-Xian Li and Benjamin M. Marlin. Learning from irregularly-sampled time series: a missing data perspective. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[151] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[152] Hansika Hewamalage, Klaus Ackermann, and C. Bergmeir. Forecast evaluation for data scientists: common pitfalls and best practices. *Data Mining and Knowledge Discovery*, 37:788 – 832, 2022.

[153] David H. Wolpert. *The Supervised Learning No-Free-Lunch Theorems*, pages 25–42. Springer London, London, 2002.

# Glossary

**AdaBoost** Adaptive Boosting Classification/Regression Algorithm.

**ADWIN** Adaptive Windowing Drift Detector.

**AI** Artificial Intelligence.

**AIOps** Artificial Intelligence for Information Technology Operations.

**CPU** Central Processing Unit.

**DDB** Data Distribution-Based.

**DDM** Drift Detection Method Drift Detector.

**EDDM** Early Drift Detection Method Drift Detector.

**EDE** Equal Density Estimation Drift Detector.

**ELC** Effectiveness per Unit of Labeling Cost.

**ERB** Error Rate-Based.

**ERC** Effectiveness per Unit of Retraining Cost.

**ETFE** Entropy-Based Time Domain Feature Extraction Drift Detector.

**EWMA** Exponentially Weighted Moving Average.

**FEDD** Feature Extraction Drift Detector.

**FFT** Fast Fourier Transform.

**FH** Full History.

**FI** Feature Importance.

**FPR** False Positive Rate.

**GPU** Graphics Processing Unit.

**HDDM_A** Hoeffding's Inequality Carried with A-test Drift Detector.

**HDDM_W** Hoeffding's Inequality Carried with W-test Drift Detector.

**kdqTrees** Quad-Trees which Scale with the Size (k) and Dimensionality (d) of the Data Drift Detector.

**KL Divergence** Kullback–Leibler Divergence.

**KPI** Key Performance Indicator.

**KS** Kolmogorov–Smirnov Statistical Test.

**KS-ALL** Kolmogorov–Smirnov Statistical Test Applied on All Avaliable Features.

**KS-FI** Kolmogorov–Smirnov Statistical Test Applied on Features with High Feature Importance Ranking.

**KS-PCA** Kolmogorov–Smirnov Statistical Test Applied on Features Resulted from Principal Component Analysis.

**L** Latency.

**LightGBM** Light Gradient Boosting Machine Classification/Regression Algorithm.

**LSTM-AE** Long Short-Term Memory Autoencoder.

**MASE** Mean Absolute Scaled Error.

**MDI** Mean Decrease in Impurity.

**MDP** Miss-Detection Probability.

**ML** Machine Learning.

**PCA** Principal Component Analysis.

**PCA-kdq** Quad-Trees which Scale with the Size (k) and Dimensionality (d) of the Data with Dimensionality Reduced Through Principal Component Analysis Drift Detector.

**PCI** Prediction Confidence Interval.

**ROC AUC** Receiver Operating Characteristic – Area Under the Curve.

**RS** Retraining Savings.

**SMART** Self-Monitoring, Analysis, and Reporting Technology.

**SR** Spectral Residuals.

**SR-CNN** Spectral Residuals Convolutional Neural Networks.

**SW** Sliding Window.

**TNR** True Negative Rate.

**TPR** True Positive Rate.

**XGBoost** Extreme Gradient Boosting Classification/Regression Algorithm.

# Curriculum Vitæ

## Lorena Poenaru-Olaru

1995/11/16 Date of birth in Craiova, Romania

## Professional Experience

2024-Present Data Scientist, ING Netherlands

2021-2024 Machine Learning Researcher, ING Netherlands

2019-2020 Intern Data Scientist, Exact Netherlands

2017-2018 Software Engineer, Infineon Technologies Romania

2016-2016 Intern Software Engineer, Kepler Rominfo Romania

## Education

2020-2026 Doctor of Philosophy (PhD), Computer Science, Delft University of Technology, Netherlands.
**PhD Thesis**: Monitoring and Maintaining Machine Learning Models Against Concept Drift in the Context of AIOps Systems
Supervisors: Dr. Luis Cruz, Prof. Dr. Jan S. Rellermeyer and Prof. Dr. Arie van Deursen

2018-2020 Master of Science (MSc), Computer Science - Data Science and Technology, Delft University of Technology, Netherlands.
**Master Thesis**: Credit Scoring Prediction using Network-based Features. A machine learning based tool that predict the default (bankruptcy) of Dutch small or medium enterprises (SMEs). This project was conducted in collaboration with Exact Netherlands. Supervisors: Dr. Huijuan Wang and Dr. Judith Redi

2014-2018 Bachelor of Science (BSc), Computer Science - Computers and Information Technology, University "Politehnica" of Bucharest.
**Bachelor Thesis**: Lifestyle Rec - an Android application that uses collaborative filtering algorithms to recommend users' personalized relaxing activities to mitigate daily stress. This project was conducted within the CAMPUS Research Center Bucharest. Supervisor: Dr. Anamaria Rădoi

## Industry Talks during PhD

2022 *Automating Machine Learning Model Maintenance and Monitoring* - presented @Data Science Community Conference within **ING Netherlands**

2023 *Monitoring and Reacting to Concept Drift in Deployed Machine Learning* - presented @Data Science Talks Series within **Booking.com Netherlands**

2023 *Monitoring and Maintaining Anomaly Detection Models in Production* - presented @Data Science Community Conference within **ING Netherlands**

2024 *Towards Understanding the Behavior of Deployed Models Over Time: A Study Case on AIOps* - presented @MLOps Community Meetup Amsterdam

2024 *Handling Large Complex Software using AIOps Techniques* - presented @Data Science Guild within **Exact Netherlands**

# List of Publications

Included in this thesis.

1. **Lorena Poenaru-Olaru**. *AutoML: towards automation of machine learning systems maintainability.* 22nd International Middleware Conference: Doctoral Symposium (Middleware 2021). Association for Computing Machinery. https://doi.org/10.1145/3491087.3493674.

2. **Lorena Poenaru-Olaru**, Judith Redi, Arthur Hovanesyan and Huijuan Wang. *Default Prediction Using Network Based Features.* Complex Networks & Their Applications X. COMPLEX NETWORKS 2021. Studies in Computational Intelligence, vol 1072. Springer, Cham. https://doi.org/10.1007/978-3-030-93409-5_60.

3. **Lorena Poenaru-Olaru**, Luis Cruz, Arie van Deursen and Jan S. Rellermeyer. *Are Concept Drift Detectors Reliable Alarming Systems? - A Comparative Study.* IEEE International Conference on Big Data (Big Data), Osaka, Japan, 2022, pp. 3364-3373, https://doi.org/10.1109/BigData55660.2022.10020292.

4. **Lorena Poenaru-Olaru**, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Maintaining and Monitoring AIOps Models Against Concept Drift.* IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN), Melbourne, Australia, 2023, pp. 98-99, https://doi.org/10.1109/CAIN58948.2023.00024.

5. **Lorena Poenaru-Olaru**, June Sallou, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Retrain AI Systems Responsibly! Use Sustainable Concept Drift Adaptation Techniques.* IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS), Melbourne, Australia, 2023, pp. 17-18, https://doi.org/10.1109/GREENS59328.2023.00009.

6. **Lorena Poenaru-Olaru**, Natalia Karpova, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Is Your Anomaly Detector Ready for Change? Adapting AIOps Solutions to the Real World.* IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN '24), Lisbon, Portugal, 2024, pp. 222–233, https://doi.org/10.1145/3644815.3644961.

7. **Lorena Poenaru-Olaru**, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Improving the Reliability of Failure Prediction Models through Concept Drift Monitoring.* IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest), Ottawa, Canada, 2025, pp. 1–8, https://doi.org/10.1109/DeepTest66595.2025.00006.

8. **Lorena Poenaru-Olaru**, June Sallou, Luis Cruz, Jan S. Rellermeyer and Arie van Deursen. *Sustainable Machine Learning Retraining: Optimizing Energy Efficiency Without Compromising Accuracy.* 11th International Conference on ICT for Sustainability (ICT4S), Dublin, Ireland, 2025, pp. 100-111, https://doi.org/10.1109/ICT4S68164.2025.00019.

9. **Lorena Poenaru-Olaru**, Wouter van 't Hof, Adrian Stańdo, Arkadiusz P. Trawiński, Eileen Kapel, Jan S. Rellermeyer, Luis Cruz and Arie van Deursen. *Prepared for the Unknown: Adapting AIOps Capacity Forecasting Models to Data Changes.* IEEE 36th IEEE International Symposium on Software Reliability Engineering (ISSRE), Sao Paulo, Brazil, 2025.

Successful industry-academia collaborations are built on five key pillars: *visibility* creates opportunities, opportunities spark *networking*, networking fosters *communication*, communication builds *trust*, and trust empowers true *collaboration*. Together, these pillars form a strong foundation that drives innovation and advances society.