FEDERAL UNIVERSITY OF PARÁ
INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING GRADUATE PROGRAM

Online Learning for Software Defect Prediction

Douglas Almeida Vidal

DM 02/2025

UFPA / ITEC / PPGEE
Guamá University Campus
Belém-Pará-Brazil
2025

FEDERAL UNIVERSITY OF PARÁ
INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING GRADUATE PROGRAM

Douglas Almeida Vidal

Online Learning for Software Defect Prediction

Dissertation submitted to the Examining Board of the Electrical Engineering Graduate Program at UFPA to obtain the Master's Degree in Electrical Engineering in the Area of Applied Computing.

Orientador: Prof. Dr. Glauco Estácio Gonçalves

Coorientador: Prof. Dr. Marcos César da Rocha Seruffo

UFPA / ITEC / PPGEE
Guamá University Campus
Belém-Pará-Brazil
2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

A447o Almeida Vidal, Douglas.
Online Learning for Software Defect Prediction / Douglas Almeida Vidal. — 2025.
79 f. : il. color.

Orientador(a): Prof. Dr. Glauco Estácio Gonçalves
Coorientador(a): Prof. Dr. Marcos César da Rocha Seruffo
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2025.

1. Just-in-Time Software Defect Prediction. 2. Online learning. 3. Semi-supervised learning. 4. Weight averaging. 5. Denoising Autoencoder. I. Título.

CDD 621.3

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Online Learning for Software Defect Prediction

AUTOR: **DOUGLAS ALMEIDA VIDAL**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM: 28/02/2025

BANCA EXAMINADORA:

Prof. Dr. Glauco Estácio Gonçalves
(Orientador - PPGEE/ ITEC/UFPA)

Prof. Dr. Marcos César da Rocha Seruffo
(Coorientador - PPGEE/ITEC/UFPA)

Prof. Dr. Aldebaro Barreto da Rocha Klautau Junior
(Avaliador Interno - PPGEE/ITEC/UFPA)

Prof. Dr. André Figueira Riker
(Avaliador Externo ao Programa - PPGCC/ICEN/UFPA)

VISTO:

Prof. Dr. Diego Lisboa Cardoso
(Coodernador do PPGEE/ITEC/UFPA)

To my parents Angela and Nelson and my brothers Ingrid and Nicolas who were my fundamental pillars throughout my entire journey.

Agradecimentos

First, I would like to thank God, who has always given me strength, wisdom and resilience to face obstacles and move forward with faith and determination.

To my family, who has always believed in me and given me unconditional support, I am eternally grateful. You are my foundation, my source of love and inspiration, and this moment is also the result of everything you have always done for me.

To my advisor, Prof. Dr. Glauco Estácio Gonçalves, I express my deepest gratitude for your guidance, patience and encouragement throughout the development of this work.

To my co-advisor, Prof. Dr. Marcos Seruffo, I would like to thank you for your valuable contribution and advice that enriched my research.

To the coordinators Prof. Dr. Rodrigo Soares, Prof. Dr. Cleviton Monteiro and scholarship holders Aline Costa, Erico André, Kauan Tavares and Marco Antônio of the Learning from Commits project, I would like to express my special thanks. The collaboration, debates and contributions of each person were fundamental to the development of this work and significantly expanded my vision and learning.

“It was destiny that brought me here, but I am the one who designed my own path.”

John Marston

Resumo

A previsão de defeitos de software Just-in-Time (JIT-SDP) busca identificar mudanças no código que podem introduzir defeitos no momento em que são realizadas, permitindo uma correção antecipada e reduzindo custos de manutenção. No entanto, modelos tradicionais de JIT-SDP enfrentam dificuldades devido ao desvio de conceito e à necessidade de grandes quantidades de dados rotulados, tornando-os menos eficazes em ambientes dinâmicos de desenvolvimento de software. Esta dissertação apresenta o modelo *Semi-supervised Stochastic Weight Averaging (S3WA)*, uma abordagem de aprendizado adaptativo que utiliza dados rotulados e não rotulados, ajustando-se dinamicamente às mudanças na distribuição dos dados. O modelo foi avaliado comparativamente a técnicas de aprendizado online de última geração em conjuntos de dados artificiais e reais, com foco especial no JIT-SDP. Os resultados mostram que o S3WA mantém uma maior precisão preditiva ao longo do tempo, lidando melhor com a desvio de conceito e reduzindo a dependência de dados rotulados. Esses achados demonstram o potencial das abordagens semissupervisionadas adaptativas para aprimorar a previsão de defeitos em tempo real em fluxos de trabalho de desenvolvimento de software.

Palavras-chave: Previsão de Defeitos de Software Just-in-time, Aprendizado Online , Aprendizado Semissupervisionado, Média de peso, Autoencoder Denoising.

Abstract

Just-in-Time Software Defect Prediction (JIT-SDP) aims to detect defect-inducing code changes at the moment they are committed, allowing developers to take proactive measures to ensure software quality. However, traditional JIT-SDP models struggle with concept drift and the need for large amounts of labeled data, making them less effective in dynamic software development environments. This master thesis introduces the Semi-supervised Stochastic Weight Averaging (S3WA) model, an adaptive learning approach that leverages both labeled and unlabeled data while dynamically adjusting to evolving data streams. The model is evaluated against state-of-the-art online learning techniques using both artificial and real-world datasets, with a particular emphasis on JIT-SDP scenarios. The results demonstrate that S3WA maintains higher predictive accuracy over time compared to existing models, effectively handling concept drift while reducing the reliance on labeled data. These findings highlight the potential of adaptive semi-supervised approaches to improve defect prediction in real-time software development workflows.

Key-words: Just-in-Time Software Defect Prediction, Online learning, Semi-supervised learning, Weight averaging, Denoising Autoencoder.

List of figures

Figura 1 – Representation of different types of concept drifts. 7
Figura 2 – Types of concept drifts in terms of time. 8
Figura 3 – Fluxogram of the JIT-SDP 21
Figura 4 – S3WA Architecture 27
Figura 5 – Illustration of the learning rate schedule adopted by SWA. 31
Figura 6 – Plots of the prequential accuracy surface depicting S3WA and SWA optimization paths during an abrupt concept drift on Sine1 data stream. 34
Figura 7 – Critical difference diagram of Nemenyi post-hoc statistical test across all data streams. Values in brackets are mean ranks. 42
Figura 8 – Plots of prequential accuracy produced by the best run of each of the methods for artificial data streams with abrupt concept drifts. Dashed lines denote concept drifts. 44
Figura 9 – Critical difference diagram of Nemenyi post-hoc statistical test on mean prequential accuracy across all data streams. Values in brackets are mean ranks. 46
Figura 10 – Computational time of numbers of base learners 48
Figura 11 – Plots of prequential accuracy produced by the best run of each of the methods. 52

List of tables

Tabela 1 – Comparision of methods 25
Tabela 2 – Summary of data streams. 38
Tabela 3 – Table of hyperparameter ranges for random search. 41
Tabela 4 – Processing time (in seconds) for each method on each data stream. 43
Tabela 5 – Mean of prequential accuracy for all methods. 47
Tabela 6 – The number of bug-inducing and clean commits in the projects of the ApacheJIT 49
Tabela 7 – Mean of prequential accuracy for all methods. 53
Tabela 8 – Processing time (in seconds) for each method on each data stream. 53
Tabela 9 – Mean of prequential accuracy for S3WA. 55

Lista de abreviaturas e siglas

ADL	Adaptive Deep Learning
ADWIN	Adaptive Windowing
AGMM	Autonomous Gaussian Mixture Mode
CISQ	Consortium for IT Software Quality
CSB2	Cost-Sensitive Boosting 2
DAE	Denoising Autoencoder
DDS	Drift Detection Scenario
DL	Deep Learning
DNN	Deep Neural Network
DWM	Dynamic Weighted Majority
DEVDAN	Deep Evolving Denoising Autoencoder
EMA	Exponential Moving Average
IoT	Internet of Things
JIT-SDP	Just-in-Time Software Prediction
ML	Machine Learning
MLP	Multilayer Perceptron
MOA	Massive Online Analysis
NS	Network Significance
OzaBagAdwin	Online Bagging
ParsNet	Parsimonious Network
SDLC	Software Development Lifecycle
SDP	Software Defect Prediction
SRPC	Streaming Random Patches Classifier

SWA	Stochastic Weight Averaging
S3WA	Semi-supervised Stochastic Weight Averaging
V&V	Verification and Validation

Sumário

1 INTRODUCTION 1
1.1 Context 1
1.2 Motivation 2
1.3 Objectives 3
1.3.1 General Objective 3
1.3.2 Specific Objectives 4
1.4 Document structure 4
2 Online Learning 5
2.1 Non-stationary Environments 5
2.2 Online Learning 9
2.3 Online Models 10
2.3.1 Supervised and Chunk Learning 10
2.3.1.1 Streaming Random Patches Classifier (SRPC) 11
2.3.1.2 Dynamic Weighted Majority (DWM) 12
2.3.1.3 Cost-Sensitive Boosting 2 (CSB2) 13
2.3.1.4 Online Bagging (OzaBagAdwin) 14
2.3.2 Semisupervised and Batch Learning 15
2.3.2.1 Parsimonious Network (ParsNet) 15
2.3.2.2 Deep Evolving Denoising Autoencoder (DEVDAN) 17
2.3.2.3 Adaptive Deep Learning (ADL) 18
3 Just-in-Time Software Defect Prediction (JIT-SDP) 20
3.1 What is JIT-SDP 20
3.2 Selected JIT-SDP work 22
4 S3WA 26
4.1 Network Architecture 26
4.2 Learning Algorithm 29
4.3 S3WA's optimization paths 31
5 RESULTS 36
5.1 Results with state-of-the-art datasets 37
5.1.1 Experimental Method 37
5.1.2 Experiments with online batch learning 40
5.1.3 Experiments with chunk based learning 45
5.2 Results with JIT-SDP datasets 48
5.2.1 Experimental Method 48
5.2.2 Experiments with online batch learning 50
5.2.3 Experiments with chunk based learning 54

5.3 Discussion 55
6 Conclusion 58
6.1 Contributions 59
6.2 Future Work 60

Referências 61

1 INTRODUCTION

1.1 Context

Ensuring high-quality software is a complex endeavor, requiring effective collaboration of different teams across planning, development, and testing phases. Despite these efforts, many software products still suffer from a high defect rate in production. These defects can lead to serious outcomes, including financial losses, missed deadlines, reputation damages, and, in safety-critical applications, risks to human lives (PACHOULY et al., 2022).

The financial impact of software errors is profound. The Consortium for IT Software Quality (CISQ) estimated that poor-quality software costs the U.S. economy $2.41 trillion in 2022, with even higher costs projected for 2024 (KRASNER, 2022). Beyond direct costs, defects reduce productivity, disrupt operations, and delay projects[1]. They can also tarnish a company's reputation, signaling weak quality standards, and introduce security vulnerabilities, increasing the likelihood of cyberattacks, data breaches, and theft.

Software testing is, therefore, essential to Verification and Validation (V&V), ensuring both functional accuracy and long-term reliability (THOTA; SHAJIN; RAJESH, 2020). However, the resources required for testing — time, budget, infrastructure, and skilled personnel — can rival those needed for development itself (LEMOS et al., 2015). This is especially critical for safety-critical systems, where failure can have severe consequences, making efficient testing strategies imperative.

The Software Development Life Cycle (SDLC) generates a wealth of historical data that offers valuable insights into improving software quality (PACHOULY et al., 2022). Tools like Jira, Confluence, Jenkins, Perforce, and GitHub help capture essential artifacts across various phases. By analyzing this data, organizations can leverage advanced technologies to optimize development workflows, improving both product quality and delivery timelines. Predicting software defects early in the SDLC is particularly effective, offering a practical alternative to labor-intensive manual inspections (WAHONO, 2015).

Software Defect Prediction (SDP) aims to identify potential issues before they escalate, helping to maintain high-quality standards. Machine Learning (ML) and Deep Learning (DL) models play a decisive role in SDP, with their ability to predict not only defects but also related factors like severity, resource allocation, and defect types (LESSMANN et al., 2008). These insights enable teams to allocate resources efficiently and make data-driven decisions.

A specialized approach to SDP — Just-in-Time Software Defect Prediction (JIT-SDP)

[1] www.forbes.com/councils/forbestechcouncil/2023/12/26/costly-code-the-price-of-software-errors/

— has gained traction in recent years. Unlike conventional methods that work at a coarse level over consolidated files or modules, JIT-SDP focuses on predicting whether a code change will introduce defects immediately after it is committed, i.e., as soon as a file or module suffers a change, even if it is a small change (KAMEI et al., 2012). This fine-grained approach offers several advantages: it reduces the effort needed to address defects, assigns fixes to the relevant developer, and ensures timely feedback while the code is still fresh in the developer's memory.

Most existing studies on JIT-SDP assume an offline context, where a static dataset is used for model training (KAMEI et al., 2016). In 2012, (KAMEI et al., 2012) used Logistic Regression model with SZZ in six outsourced and five commercial projects and (LI et al., 2020a), in 2020, develop a semi-supervised Effort-aware JIT-SDP model where used classifiers like Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF). However, real-world JIT-SDP operates in an online setting, where labeled data becomes available incrementally as software changes are committed. This dynamic environment introduces the challenge of concept drift — changes in data distribution over time (LU et al., 2018).

Concept drift in JIT-SDP can result from evolving development practices, software architecture updates, or shifts in user behavior (MCINTOSH; KAMEI, 2018). If not addressed, this drift can lead to deteriorating model performance, increasing the likelihood of false positives and negatives (DONG et al., 2017). Adaptive strategies to detect and respond to concept drifts are a cornerstone of an adaptable JIT-SDP system, responsible for maintaining accurate predictions and ensuring that defect resolution remains efficient. These strategies not only sustain the effectiveness of JIT-SDP models but also reduce post-release defect costs, ultimately enhancing software reliability and quality.

Therefore, this master thesis focuses on defect prediction within the JIT-SDP scenario, while simultaneously proposing the neural network Semi-supervised Stochastic Weight Averaging (S3WA) model as an innovative solution. To achieve this, S3WA is initially compared with other state-of-the-art online models using artificial and real datasets. Subsequently, the models are applied to JIT-SDP specific datasets to determine the most effective approach for defect prediction in this context. This study aims to enhance software quality and support decision-making by introducing a robust and adaptive model tailored to dynamic and non-stationary environments.

1.2 Motivation

The application of ML in dynamic environments, such as real-time software development, presents significant challenges due to the phenomenon of concept drift, where the statistical properties of the target variable change over time, leading to model performance degradation (HOVAKIMYAN; BRAVO, 2024). Traditional ML models, designed for stationary environments, often struggle to maintain accuracy under these conditions, resulting in increased false positives and negatives.

In the context of JIT-SDP, accurate and timely detection of potential defects is crucial to prevent failures, reduce costs, and ensure software reliability (PANDEY; JADHAV, 2024). However, existing approaches often rely heavily on fully supervised learning paradigms, which require large volumes of labeled data for effective training (ZHANG; JING; WANG, 2017). Labeling data is frequently costly, time-consuming, and impractical, especially in real-time environments where data is continuously generated. To address these limitations, semi-supervised learning techniques have been proposed, allowing models to leverage both labeled and unlabeled data, thereby reducing the dependency on large labeled datasets.

Additionally, adaptive strategies that can respond to concept drift more effectively are being explored to maintain high predictive accuracy in non-stationary environments. Despite advancements in integrating semi-supervised learning and adaptive strategies, challenges remain in balancing computational efficiency and predictive performance (HOVAKIMYAN; BRAVO, 2024). Developing robust frameworks for JIT-SDP has significant implications, including more efficient allocation of development resources, quicker resolution of potential defects, and substantial cost savings, particularly in large-scale or safety-critical software systems (PANDEY; JADHAV, 2024).

This master thesis is motivated by the need to bridge this gap by proposing innovative solutions that enhance defect prediction accuracy while minimizing the complexities associated with maintaining adaptive models. With this, the S3WA model is introduced as a promising solution. By leveraging semi-supervised learning techniques, S3WA reduces the dependence on labeled data, making it better suited for real-time applications. Its adaptive design also helps it respond to concept drift more effectively, maintaining high prediction accuracy in non-stationary environments.

The development of a robust framework for JIT-SDP has long implications. It can lead to more efficient allocation of development resources, faster resolution of potential defects, and significant cost savings, especially in large-scale or safety-critical software systems. By introducing adaptive solutions, this work aims to not only address current limitations but also set a foundation for future advancements in dynamic software quality assurance.

1.3 Objectives

1.3.1 General Objective

This master thesis proposes the S3WA algorithm, integrating semi-supervised learning techniques and Stochastic Weight Averaging (SWA) to handle non-stationary online learning in the JIT-SDP, comparing it with state-of-the-art models and demonstrating which model performs better in predicting software defects.

1.3.2 Specific Objectives

To achieve the general objective, the following specific objectives are established:

- To Investigate the main challenges related to the use of online models in JIT-SDP, with emphasis on dynamic scenarios and concept drift;
- Develop the S3WA algorithm, integrating semi-supervised learning techniques and SWA to handle non-stationary data streams;
- Design and develop an experimental framework to systematically assess the performance of S3WA and other state-of-the-art models, focusing on their stability, accuracy, and adaptability across various scenarios;
- Perform a detailed comparison between S3WA and other existing online models in the literature, highlighting their advantages and limitations across different scenarios;
- Assess the performance of S3WA and online models in the context of JIT-SDP, identifying the most suitable model for defect prediction in dynamic scenarios.

1.4 Document structure

This master thesis is structured to provide a comprehensive and systematic understanding of the application of online models in the context of JIT-SDP. To ensure clarity and depth, the work is organized into the following chapters:

Chapter 2 explores the core concepts of supervised and semi-supervised learning, non-stationary environments, concept drift, and online models. It provides the theoretical foundation for understanding online learning strategies applicable to dynamic online learning.

Chapter 3 discusses the principles of JIT-SDP, its challenges, and its significance in software development. It highlights how JIT-SDP integrates with online models to predict software defects effectively.

Chapter 4 details the methodology, including the design and implementation of the S3WA algorithm. It describes how S3WA integrates semi-supervised learning with stochastic weight averaging to handle non-stationary data streams.

Chapter 5 interprets the obtained results, critically discussing the performance of online models, including S3WA, in non-stationary datasets and JIT-SDP scenarios.

Finally, Chapter 6, summarizes the research contributions and suggests directions for future work.

2 Online Learning

The ever-growing volume of data generated in real-time by modern systems, such as the Internet of Things (IoT), financial markets, and software repositories, has made traditional batch learning models increasingly inadequate. These models, which require static datasets for training, struggle to adapt to dynamically changing environments. In contrast, online learning has emerged as a crucial approach for addressing the challenges of real-time data processing, offering adaptive mechanisms that continuously update models as new data arrives.

This section explores the fundamental concepts and challenges associated with online learning in non-stationary environments. We begin by discussing the impact of data streams and concept drift, emphasizing how traditional learning paradigms fail to cope with evolving data distributions (Section 2.1). Next, in Section 2.2, we present key online learning techniques, highlighting their ability to adjust to dynamic changes while maintaining predictive accuracy.

2.1 Non-stationary Environments

The extensive adoption of mobile devices, the evolution of the IoT, and the expansion of sensor networks have resulted in an unprecedented and continuously growing volume of data, often delivered in a streaming manner (ZHOU et al., 2014). A common assumption, whether implicit or explicit, is that the process generating this data stream is stationary. This implies that the data is sampled from a fixed, albeit unknown, probability distribution (DITZLER et al., 2015). However, in numerous real-world applications, this assumption is invalid, as the underlying data-generating process is inherently dynamic, exhibiting non-stationary behavior (often termed evolution or drift). Non-stationarity can emerge due to various factors, including seasonal or cyclical patterns, shifts in user behavior or preferences, malfunctions in hardware or software within cyber-physical systems, thermal fluctuations, or sensor degradation over time.

In non-stationary environments, where the statistical properties of the data evolve over time, a model trained under the incorrect assumption of stationarity will inevitably become outdated, leading to subpar performance or even critical failures (DITZLER et al., 2015). These issues are further amplified in scenarios where data is continuously generated, and models are required to make decisions in real time. Without mechanisms to identify and adapt to these changes, models are prone to significant performance deterioration. This can result in higher error rates, diminished reliability in decision-making, and, in severe cases, an inability to generalize to new data patterns.

The primary challenges faced by machine learning models in non-stationary environments include Data Scarcity and Imbalance, Real-Time Adaptation, Model Evaluation and

Validation, and Concept Drift.

Data scarcity and data imbalance are critical challenges in learning from non-stationary data streams, directly impacting a model's ability to adapt and generalize (LI et al., 2020b). Data scarcity occurs when there are too few examples available for effective model training, making it difficult to capture representative patterns and increasing vulnerability to overfitting, especially in dynamic environments where patterns may become rare or infrequent. On the other hand, data imbalance arises when there is a disproportionate distribution between classes, causing the model to favor the majority class—a problem that worsens with concept drifts, where the relative frequency of classes may change over time. The combination of scarcity and imbalance further complicates adaptive learning, as traditional models that assume stable distributions often fail to recognize and adjust to these changes. Strategies such as resampling, semi-supervised learning, and dynamic ensemble methods can help mitigate these effects, enabling models to maintain robust predictive performance even in highly variable scenarios.

Real-time adaptation occurs when models must be continuously or periodically updated to incorporate new data patterns while operating under memory constraints. Additionally, non-stationary environments often demand real-time or near-real-time adaptation, making efficient online approaches, particularly those deployable on edge devices, highly relevant (CHEN et al., 2021). Real-time adaptation is crucial in scenarios where data patterns change dynamically over time, such as in monitoring systems, IoT devices, and financial applications. Without this capability, models trained on past distributions quickly become obsolete, reducing their effectiveness and reliability. Online learning methods allow models to gradually adjust their parameters as new data becomes available, without requiring full retraining, which would be impractical in resource-constrained environments. This characteristic is particularly important for systems deployed on edge devices, where memory and processing power are limited, necessitating techniques that are both efficient and optimized for local execution.

Model Evaluation and Validation occurs when traditional evaluation techniques may be inadequate for non-stationary environments, as they typically depend on static test datasets that fail to capture the dynamic nature of the data. In supervised online learning, evaluation often employs strategies such as interleaved train-test error assessment (LOSING; HAMMER; WERSING, 2018). In this context, model evaluation refers to the process of assessing the predictive performance of a learning algorithm, often using metrics such as accuracy, precision, recall, and F1-score. However, in non-stationary environments, where data distributions evolve over time, static evaluation methods become unreliable, as they do not reflect the model's ability to adapt to concept drifts. Instead, online learning techniques rely on dynamic evaluation strategies, such as prequential evaluation. In this approach, predictions are made sequentially as new data arrives, and their correctness is assessed before the instance is used for training. This method ensures a continuous and realistic assessment of the model's performance in streaming environments. A common variant, prequential accuracy with a forgetting factor, assigns greater

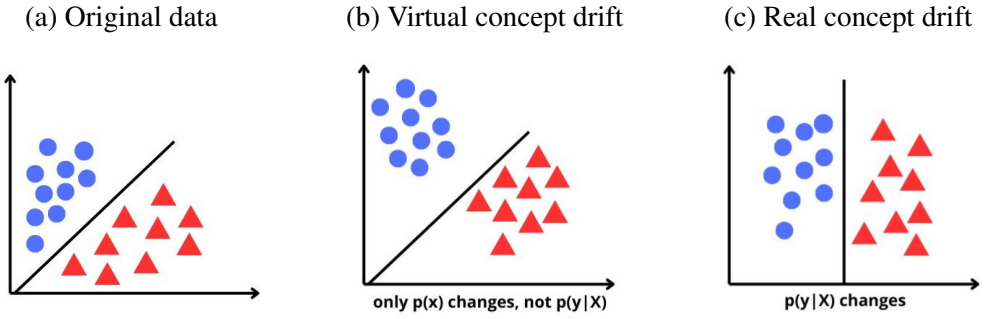
importance to recent observations, making it particularly effective in non-stationary data streams (GAMA; SEBASTIAO; RODRIGUES, 2013).

Furthermore, validation in non-stationary scenarios often requires mechanisms to detect and react to changes in the data distribution, ensuring that the model remains relevant despite evolving patterns. This highlights the necessity of adaptive evaluation frameworks that can accommodate gradual, abrupt, and recurrent shifts in data, ultimately improving the robustness of models deployed in real-world applications.

A phenomenon referred to as concept drift describes a situation where the statistical properties of a target class change unpredictably over time (PALLI et al., 2024). These changes may arise due to shifts in factors that are difficult to quantify or identify explicitly. Formally, learning in such scenarios depends on an underlying family of distributions D_t, where t represents the current time point. Concept drift occurs when $D_{t_1} \neq D_{t_2}$ for at least two distinct time points t_1 and t_2. This drift can manifest as alterations in the input distribution, the model's predictions, or the feature representation (HINDER; VAQUET; HAMMER, 2024). Consequently, the existing model may become ineffective, either because no single model can adequately represent both D_{t_1} and D_{t_2} or because the variability in D_{t_2} cannot be accurately predicted based on D_{t_1}.

Concept drift is primarily categorized into two types: virtual drift and real drift, as depicted in Figure 1 (AGRAHARI; SINGH, 2022). Virtual drift can be understood as a change in the conditional probabilities $P(x)$, where the shift occurs only in the data distribution without altering the class boundaries. In the context of evolving data streams, if a new feature emerges or becomes more prominent within a specific class, it can lead to a change in the data distribution for that class. This shift may impact the model's ability to correctly classify instances within that class based on the existing feature set. On the other hand, real drift involves a change in the unconditional probability distribution $P(y|x)$, which can affect the relationships between features and labels, potentially altering the class boundaries learned by the model.

Figure 1 – Representation of different types of concept drifts.



Source: Author

Real drift can occur due to shifts in environmental conditions, user behavior, or other external factors. The performance of a model may be influenced by both types of drift—virtual

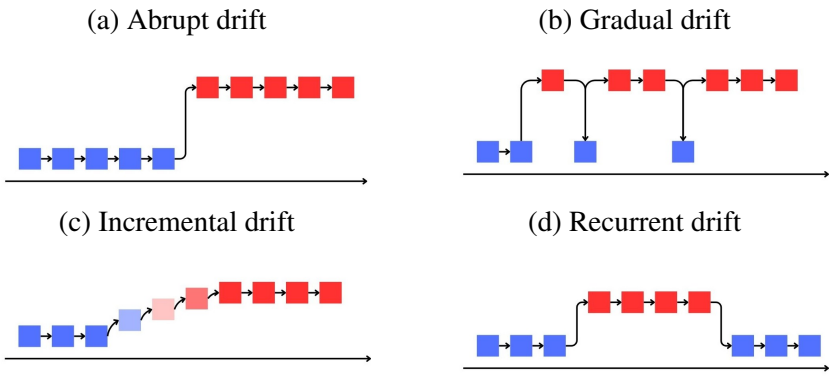
and real—either simultaneously or in distinct ways.

In terms of temporal dynamics, concept drift manifests in four primary forms: abrupt drift, gradual drift, incremental drift, and recurrent drift (KRAWCZYK; CANO, 2018).

Figure 2 illustrates these types of concept drift in relation to time. In Figure 2a, abrupt drift occurs when the new concept in the incoming data stream suddenly replaces the old one. This is typically observed when the classifier's accuracy drops sharply. Following this, the model quickly adapts to the new concept's characteristics, incorporating the changes. In gradual drift, shown in Figure 2b, the transition between concepts takes place over a longer period compared to abrupt drift. This type of drift has two variations: slow gradual drift and normal gradual drift. For instance, gradual drift can be observed in market dynamics shifting due to inflation or recession. During this process, there is an overlap between the old and new concepts, and eventually, the new concept stabilizes.

Figure 2c depicts incremental drift, which is similar to gradual drift but occurs in small, progressive steps. Each change is subtle and almost imperceptible until the new concept is fully established. Finally, in recurrent drift, illustrated in Figure 2d, old concepts reappear after a certain period. This is common in seasonal or cyclical environments. For example, in a traffic forecasting system, holiday-related congestion patterns may reoccur annually at the same time.

Figure 2 – Types of concept drifts in terms of time.



Source: Author

The presence of concept drift in non-stationary data streams underscores the necessity for adaptive methods capable of sustaining predictive accuracy over time. As data distributions evolve, whether suddenly or incrementally, predictive models encounter substantial challenges in maintaining their performance. To address these dynamic changes, online learning techniques provide a powerful solution, enabling models to be continuously updated as new data arrives (AGRAHARI; SINGH, 2022). These approaches are particularly valuable in environments where data streams are non-stationary, as they allow models to adapt to shifting patterns without requiring complete retraining. The following section will delve into the fundamentals of online learning, examining its benefits and limitations in the context of non-stationary environments.

2.2 Online Learning

Online learning is a process that involves answering a sequence of questions based on partial or complete knowledge of the correct answers to previous questions, along with any additional available information (SHALEV-SHWARTZ et al., 2012). Unlike traditional batch learning where requires the entire dataset to be available upfront for training, making it suitable for static data scenarios but limited in dynamic environments as it necessitates periodic retraining to incorporate new information. Chunk learning processes data incrementally by dividing it into blocks (chunks) that update the model as new data arrives, allowing better adaptation to changes in the data stream without full retraining. A common extension of this approach is chunk learning with a sliding window, where only the most recent chunks are retained for training while older data is discarded, ensuring that the model adapts to new patterns while mitigating the risk of outdated information affecting predictions (BARTZ-BEIELSTEIN, 2024). This approach enables models to stay relevant and accurate without the need for retraining from scratch, making it computationally efficient and highly responsive to evolving data patterns.

In online learning, the process unfolds over a series of consecutive rounds. At each round t, the learner is presented with a question, x_t, drawn from an instance domain X, and is required to provide an answer, denoted as p_t. After making the prediction, the correct answer, y_t, from a target domain Y, is revealed. The learner then incurs a loss, $l(p_t, y_t)$, which quantifies the discrepancy between the predicted answer and the correct one. While in many cases p_t belongs to Y, it is sometimes convenient to allow the learner to choose a prediction from a larger set, denoted as D. A specific case of this work, where both predictions and answers are binary (i.e., $D = Y = \{0, 1\}$), is referred to as online classification. In this scenario, the 0–1 loss function is commonly used: $l(p_t, y_t) = |p_t - y_t|$. This loss function simply indicates whether the prediction was correct ($p_t = y_t$) or incorrect ($p_t \neq y_t$).

In this setting, the concept drift detection agent observes a continuous and undifferentiated stream of incoming events and must classify each event as either anomalous or normal (LANE; BRODLEY, 1998). This learning task presents several challenges not encountered in traditional, static learning environments. One of the most significant challenges in online learning is handling non-stationary environments, where the underlying data distribution changes over time. This phenomenon, known as concept drift, can take various forms (as discussed in Section 2.1). Effective online learning systems must be capable of detecting and adapting to such drifts to maintain predictive accuracy. Techniques like Page-Hinkley test (SEBASTIÃO; FERNANDES, 2017), CUSUM (KADWE; SURYAWANSHI, 2015), or sliding window-based methods (FERREIRA; RUANO, 2009) are often employed to detect changes in data distribution. Once a drift is detected, strategies such as reweighting model parameters or selectively updating the model can be used to adapt to new conditions. In more complex scenarios, where multiple drifts occur simultaneously (e.g., abrupt and gradual changes), hybrid approaches that combine detection and continuous adaptation are essential to maintain model performance.

Practical applications of online learning are widespread, particularly in fields where timely and adaptive decision-making is critical. For example, in anomaly detection, models are used to monitor network traffic or user activity to identify deviations from normal patterns (HAN et al., 2021). In financial markets, online learning enables dynamic prediction of stock prices and detection of fraudulent transactions (ZHANG; WU; LI, 2022). Another important extension is semi-supervised online learning, which leverages both labeled and unlabeled data. This approach is particularly beneficial in scenarios where labeled data is scarce or expensive to obtain. By effectively incorporating unlabeled data, models can improve performance and reduce dependence on extensive labeled datasets. Techniques like auto-labeling (PRATAMA; ASHFAHANI; HADY, 2019) are often used in semi-supervised online learning to enhance model accuracy.

To facilitate the implementation of online learning algorithms, several frameworks and tools are available. For instance, River is a Python library specifically designed for online learning on data streams, while Massive Online Analysis (MOA) offers a variety of algorithms for real-time data analysis. Integration with big data platforms like Apache Flink and Apache Kafka enables efficient processing of large-scale data streams, making online learning feasible for industrial and commercial applications.

In summary, online learning provides a powerful framework for adapting to changing data distributions, making it indispensable in applications ranging from anomaly detection to financial markets and beyond. By leveraging techniques like semi-supervised learning, drift detection, and adaptive algorithms, online learning systems can maintain high performance even in non-stationary environments.

2.3 Online Models

Online models provide rapid adaptation to new information, ensuring consistent performance even in non-stationary scenarios. They employ different strategies, such as adaptive ensembles and self-evolving neural networks, to detect and respond to changes in data, making them indispensable tools for applications in data streams. In this section, we will discuss in detail some of the main online models, analyzing their characteristics and mechanisms for adapting to the complexities of constantly evolving data streams.

2.3.1 Supervised and Chunk Learning

Supervised learning, widely applied across various online learning scenarios, relies on the continuous availability of labeled data to train predictive models (JAIN et al., 2014). In dynamic data stream environments, these models face unique challenges, such as rapid adaptation to changing data patterns and maintaining predictive performance in real-time. Chunk learning complements these approaches by grouping data into manageable segments, enabling

incremental model training and evaluation (ESHKOL-TARAVELLA et al., 2020). This section discusses some of the key supervised algorithms, including the Streaming Random Patches Classifier (SRPC), Dynamic Weighted Majority (DWM), Cost-Sensitive Boosting 2 (CSB2), and Online Bagging (OzaBagAdwin), highlighting their features, advantages, and limitations in non-stationary environments.

2.3.1.1 Streaming Random Patches Classifier (SRPC)

The SRPC model is an innovative approach for classifying evolving data streams, combining strategies of random subspaces and online bagging (GOMES; READ; BIFET, 2019). It is designed to address the challenges of data streams, such as high real-time data volume and the need to adapt to continuous changes in data patterns, known as concept drift. SRPC uses random patches, which consist of subsets of instances and attributes extracted from the data stream, to train each base model in the ensemble. This combination enhances diversity among the models and improves predictive performance compared to traditional ensemble methods.

Each base model in SRPC is trained using random patches, defined as simultaneous subsets of attributes and instances. The patches are determined by the parameters p_s which controls the proportion of instances selected, and p_f which defines the proportion of attributes used. Formally, the patches are extracted from the full data space S where $R(p_s, p_f, S)$ represents the set of all possible patches with sizes proportional to p_s and p_f. This strategy allows SRPC to reduce the dimensionality of the data processed by each base model, resulting in significant computational efficiency gains and greater robustness against overfitting.

Diversity among the base models is promoted through two main strategies: random sampling of instances and selection of random subspaces of attributes. Instance sampling uses the Poisson distribution, where each instance is assigned a random weight $k \sim Poisson(\lambda)$. The parameter λ controls the frequency with which each instance is used in training. Additionally, the selection of random subspaces of attributes reduces the dimensionality of the feature space, enabling the models to explore different attribute combinations. This process is theoretically supported by properties such as the Johnson-Lindenstrauss lemma, which guarantees that projection into random subspaces approximately preserves the Euclidean geometry of the data, a property beneficial for nearest-neighbor-based algorithms and decision trees.

Another key aspect of SRPC is its ability to detect and recover from concept drifts. To this end, SRPC integrates the Adaptive Windowing (ADWIN) algorithm, which continuously monitors the predictive performance of the base models. When ADWIN detects a statistically significant change in the data distribution, it signals the need to replace or update base models. This mechanism is complemented by a strategy where new models are trained in the background during the warning period. If the drift is confirmed, the background model replaces the active one, ensuring that the ensemble continues to operate with high accuracy even in the face of abrupt or gradual data changes.

Despite its advantages, SRPC is not without limitations that should be considered in its use and future development. One of SRPC main drawbacks is its dependency on parameters controlling the patch sizes, such as p_s and p_f, and the parameter λ. used for instance sampling. Poorly chosen values for these parameters can negatively impact the model's accuracy and efficiency, requiring fine-tuning and, in some cases, extensive validation, which is not always feasible in continuous data streams where concepts can change rapidly. Another disadvantage is related to cumulative computational costs. Although SRP is efficient in processing each base model, the use of an ensemble with multiple models and continuous training on data patches can significantly increase memory and CPU usage, particularly in scenarios with high-frequency data streams or a large number of attributes. This limitation becomes more pronounced when λ and p_f are high, leading to frequent training of models with many attributes, thereby consuming more computational resources and execution time. Despite these challenges, the benefits of SRP often outweigh its limitations, making it a robust and versatile solution for a wide range of continuous learning applications.

2.3.1.2 Dynamic Weighted Majority (DWM)

The DWM is an ensemble method explicitly designed to address the challenge of concept drift in online learning tasks (KOLTER; MALOOF, 2007). DWM builds upon the traditional weighted majority algorithm by introducing mechanisms to dynamically add and remove base learners, or based on their performance, thereby enabling the ensemble to adapt to non-stationary environments.

DWM operates by maintaining a pool of weighted experts, each trained using an online learning algorithm. Initially, the ensemble starts with a single expert assigned a default weight. As new data instances arrive, the ensemble generates predictions by computing a weighted vote across all experts, where each expert's weight reflects its past predictive performance. When an expert makes an incorrect prediction, its weight is reduced by a multiplicative decay factor, typically denoted as β and set to values such as 0.5. This approach ensures that poorly performing experts have a diminishing influence on the ensemble's overall decision-making process.

A crucial feature of DWM is its ability to adapt to changing data distributions by dynamically modifying the ensemble composition. Whenever the ensemble's global prediction is incorrect, a new expert is created and added to the ensemble with an initial weight. This mechanism allows the model to quickly incorporate fresh learners better suited to the current concept. To prevent the ensemble from growing indefinitely, DWM employs a pruning mechanism that removes experts whose weights fall below a predefined threshold θ. This ensures that the ensemble remains computationally efficient while focusing on high-performing learners.

Another important aspect of DWM is its normalization of expert weights. After adjusting the weights based on prediction performance, the algorithm rescales them so that the maximum weight remains one. This normalization step prevents newly added experts from

disproportionately influencing the ensemble, promoting stability in the decision-making process.

DWM is a general algorithm that can utilize any online learning algorithm as its base learner. Common choices include Naive Bayes and incremental decision tree methods, such as the Incremental Tree Inducer (ITI). The choice of base learner can significantly impact the ensemble's performance, as each algorithm has different strengths in handling specific types of concept drift. For example, Naive Bayes excels in scenarios with strong independence assumptions between features, while ITI adapts well to complex decision boundaries by dynamically restructuring its trees.

Despite its strengths, DWM has certain limitations. The algorithm's reliance on parameters, such as β, θ, and the frequency of ensemble updates, requires careful tuning to achieve optimal performance. Incorrect parameter settings can lead to inefficiencies, such as the creation of unnecessary experts or delayed adaptation to new concepts. Furthermore, the computational cost of maintaining and querying a large ensemble can become significant in scenarios with high-frequency data streams or noisy environments. However, strategies such as limiting the maximum number of experts in the ensemble or distributing computations across multiple processors can mitigate these challenges.

In conclusion, DWM represents a robust and flexible approach to handling concept drift in online learning environments. Its dynamic mechanisms for adjusting the ensemble composition, combined with effective weight management and pruning strategies, enable it to maintain high predictive accuracy in evolving data streams.

2.3.1.3 Cost-Sensitive Boosting 2 (CSB2)

The CSB2 algorithm is an ensemble method that adapts boosting techniques to address the challenge of learning in imbalanced class scenarios (WANG; PINEAU, 2016). This problem is particularly critical in applications where the minority class holds greater practical significance, such as medical diagnosis, fraud detection, and other cases where classification errors for positive or negative examples carry significantly different costs. CSB2 is designed to tackle this issue by dynamically adjusting the weights of training examples of data based on their misclassification costs, ensuring heightened sensitivity to cost-asymmetric scenarios.

Building upon AdaBoost, CSB2 modifies the weight-updating process for training examples in each iteration. Instead of treating all examples uniformly, the algorithm assigns higher weights to examples with higher misclassification costs. Specifically, CSB2 uses two key parameters: C_p representing the cost associated with false negatives (misclassifying a positive example as negative), and C_n representing the cost of false positives (misclassifying a negative example as positive). Typically, C_p is higher, reflecting the greater severity of missing a critical positive example.

During each iteration, CSB2 adjusts the weights of training examples based on the

model's performance. Correctly classified examples have their weights reduced, but the reduction differs between positive and negative examples according to their respective costs C_p and C_n. Conversely, the weights of misclassified examples are increased proportionally to their misclassification costs. This process is formalized through a modification of AdaBoost's weight-update equation, integrating C_p and C_n directly into the Formula 2.1:

$$D_{m+1}(n) = D_m(n) * C_n * exp(-a_m(2I(h_m(x_n) = y_n) - 1)) \tag{2.1}$$

where C_n represents the cost associated with each example, a_m is the weight of the classifier in iteration m and I is an indicator function that equals 1 for correct classifications and -1 for incorrect ones.

CSB2s strategy combines AdaBoost's robustness with cost sensitivity, enabling the algorithm to prioritize minority class examples or those with higher error costs. Additionally, the weights of the classifiers in the ensemble are adjusted based on their ability to minimize total classification costs, further enhancing the ensemble's effectiveness in imbalanced scenarios.

Despite its significant advantages, CSB2 is not without limitations. The algorithm's effectiveness depends on proper configuration of the C_p and C_n parameters, which may vary depending on the application and require careful tuning. Additionally, the computational cost of CSB2 can be higher than simpler methods due to the need to calculate and update weights in a cost-sensitive manner at each iteration. Nevertheless, its ability to handle class imbalance and unequal costs makes it a highly recommended choice for critical applications where the accuracy of the minority class is paramount.

2.3.1.4 Online Bagging (OzaBagAdwin)

The OzaBagAdwin algorithm is an extension of the original OzaBag method, which itself is an online adaptation of bagging, specifically designed to handle evolving data streams (BIFET et al., 2009). OzaBagAdwin integrates the concept of bagging with the ADWIN (Adaptive Windowing) algorithm to effectively address concept drift in dynamic environments. By combining these two techniques, OzaBagAdwin achieves both adaptability to changes in the data distribution and robust predictive performance in non-stationary scenarios.

OzaBag, the foundational algorithm, is based on the principles of bagging but modifies the process to work in an online setting. In traditional bagging, multiple base models are trained on different bootstrapped subsets of the dataset. OzaBag achieves this in an incremental fashion by assigning weights to each incoming data instance based on a Poisson distribution. For a given instance, the number of times it is used to train a model in the ensemble is determined by drawing a random value $k \sim Poisson(\lambda)$, where λ = 1. This process ensures diversity among the base models while allowing for continuous training as new data arrives.

OzaBagAdwin enhances this method by incorporating the ADWIN algorithm, which serves as a change detector and performance estimator. ADWIN monitors the data stream and

dynamically adjusts the ensemble in response to concept drift. Specifically, ADWIN maintains a sliding window of data and uses statistical tests to detect significant changes in the data distribution. When a change is detected, OzaBagAdwin reacts by updating the ensemble. Underperforming models are removed, and new models are added to replace them, ensuring that the ensemble remains accurate and relevant to the current concept.

The integration of ADWIN in OzaBagAdwin provides several key advantages. First, it allows the algorithm to automatically adjust to concept drift without requiring explicit drift detection mechanisms. Second, by focusing on maintaining only the most relevant models in the ensemble, OzaBagAdwin reduces the risk of overfitting to outdated data. Third, the use of ADWIN enables the algorithm to balance adaptability and stability, as it can respond to both sudden and gradual changes in the data stream.

Despite its advantages, OzaBagAdwin is not without limitations. The computational cost of maintaining and updating the ADWIN structure, particularly in high-frequency or high-dimensional data streams, can be significant. Additionally, the algorithm's performance heavily depends on the proper tuning of ADWIN's parameters, such as the sensitivity threshold for detecting changes. If these parameters are not set appropriately, the algorithm may either fail to detect concept drift in time or trigger unnecessary updates, leading to inefficiencies.

In conclusion, OzaBagAdwin is a powerful tool for learning from data streams with concept drift. By integrating the strengths of OzaBag and ADWIN, it offers a robust and adaptive approach to online learning.

2.3.2 Semisupervised and Batch Learning

Semisupervised learning emerges as a promising approach to address scenarios where labeled data is scarce or sparsely available. By combining labeled and unlabeled data, these methods maximize the use of available information, reducing dependence on labeled data and the costs associated with manual labeling. In the context of batch learning, these models tackle the challenges posed by dynamic environments and concept drift, offering adaptive and scalable solutions (BISONG; BISONG, 2019). This section explores modern frameworks such as the Parsimonious Network (ParsNet), Deep Evolving Denoising Autoencoder (DEVDAN), and Adaptive Deep Learning (ADL), highlighting their innovations, adaptive capabilities, and practical applications.

2.3.2.1 Parsimonious Network (ParsNet)

The ParsNet is a weakly supervised deep learning framework developed to address the challenges of learning from streaming data under conditions of limited or delayed label availability (PRATAMA; ASHFAHANI; HADY, 2019). ParsNet is specifically designed to manage scenarios where the ground truth labels are either sporadically accessible or available

only during an initial phase, and it effectively handles issues such as concept drift, noisy pseudo-labels, and high-dimensional data streams. By integrating a self-evolving structure and a novel self-labelling strategy called Self-Labelling with Hedge (SLASH), ParsNet demonstrates exceptional adaptability and robustness in weakly supervised learning environments.

ParsNet's architecture consists of two interconnected phases: generative and discriminative. The generative phase, built on a denoising autoencoder, focuses on extracting robust features from the data by introducing masking noise to partially corrupt input features. This phase serves not only to regularize the learning process but also to initialize the discriminative phase, which leverages these features for classification tasks. Both phases share network parameters, ensuring efficiency and consistency across the learning process. The structural evolution of ParsNet is governed by the Network Significance (NS) method, which dynamically adjusts the network's width by adding or pruning hidden nodes. This adjustment is based on statistical metrics such as bias and variance, enabling the network to expand or contract as needed to adapt to concept drift with or without access to labels.

A critical innovation in ParsNet is its SLASH mechanism, which addresses the accumulation of noisy pseudo-labels, a common issue in weakly supervised learning. When true labels are unavailable, SLASH generates pseudo-labels for unlabelled data by evaluating the confidence of predictions from both ParsNet and an adaptative Gaussian Mixture Model (AGMM). Only samples that meet predefined confidence thresholds are assigned pseudo-labels. To mitigate the impact of errors from noisy pseudo-labels, the hedge component of SLASH employs an auto-correction mechanism that adjusts network parameters back to their optimal values when deviations occur due to incorrect pseudo-labels. This process leverages the concept of Synaptic Intelligence, which tracks the importance of network parameters to ensure that critical parameters are not forgotten during updates.

The structural evolution of ParsNet is further enhanced by AGMM, which approximates the probability density function of the input data. AGMM operates in a single-pass manner and dynamically adds or removes Gaussian components to model the data distribution. This mechanism supports the network's ability to adapt to evolving data streams by introducing new components when concept drift is detected and pruning redundant components when necessary. The integration of AGMM allows ParsNet to handle complex and non-stationary data distributions without assuming a fixed Gaussian structure, making it highly versatile for real-world applications.

ParsNet's learning process also incorporates a data augmentation strategy to enrich the representation of labelled data. By injecting small amounts of Gaussian noise into the originally labelled samples, ParsNet generates augmented samples that help reinforce the decision boundaries and improve generalization. This is particularly useful in scenarios with extreme label scarcity, as it allows the network to maximize the utility of the limited labelled data available.

Despite its strengths, ParsNet has certain limitations. The framework's performance is

sensitive to the choice of hyperparameters, such as the confidence thresholds for pseudo-labelling and the regularization terms in the hedge mechanism. Additionally, the computational complexity of maintaining and updating AGMM, particularly in high-dimensional or high-frequency data streams, can pose challenges in resource-constrained environments. Nonetheless, these limitations are outweighed by the significant advantages ParsNet offers in terms of adaptability, efficiency, and scalability in weakly supervised learning contexts.

In conclusion, ParsNet represents a significant advancement in weakly supervised deep learning for streaming environments. Its ability to dynamically evolve its structure, manage noisy pseudo-labels, and adapt to concept drift without relying heavily on labelled data makes it a powerful tool for applications such as fraud detection, real-time monitoring, and adaptive recommendation systems.

2.3.2.2 Deep Evolving Denoising Autoencoder (DEVDAN)

The DEVDAN is a novel neural network framework tailored for analyzing evolving data streams (ASHFAHANI et al., 2020). It builds upon the principles of denoising autoencoders (DAE), introducing a dynamic and adaptive structure capable of handling challenges like concept drift, limited labelled data, and high-dimensional input spaces. Unlike traditional static models, DEVDAN features an open structure that evolves both in the generative and discriminative phases, ensuring it remains effective in rapidly changing environments.

DEVDAN operates in a single-pass learning fashion, allowing it to handle streaming data without revisiting previous samples. This characteristic is particularly important for applications in real-time monitoring and large-scale data analysis. The model begins with a minimal architecture and autonomously grows or prunes hidden units based on the complexity of the data, as governed by a NS formula. This formula evaluates the trade-off between bias and variance, guiding the model to adjust its structure dynamically.

In the generative phase, DEVDAN uses the denoising autoencoder mechanism to refine the network's ability to reconstruct input data from partially corrupted versions. This process helps the model extract robust features that are less sensitive to noise and better suited for downstream tasks. The reconstruction error is a critical metric here, and the NS formula derived from it enables the model to identify underfitting or overfitting scenarios. For instance, high bias indicates underfitting, prompting the addition of hidden nodes, while high variance suggests overfitting, leading to pruning.

The discriminative phase follows the generative phase once labelled data becomes available. During this phase, DEVDAN connects the feature extractor from the generative phase to a softmax layer, enabling it to perform classification tasks. This phase also incorporates the NS formula, but with modifications to account for supervised learning objectives. The ability to seamlessly integrate generative and discriminative learning makes DEVDAN highly versatile, allowing it to function effectively even in semi-supervised settings.

DEVDAN's structural evolution is managed through statistically-driven mechanisms. Hidden nodes are added or removed based on dynamic thresholds derived from the NS formula. For instance, a new node is added if the network's bias exceeds a certain confidence level, while nodes are pruned if their contribution to the variance is minimal. This ensures that the model maintains an optimal balance between complexity and generalization, adapting its architecture to the underlying data distribution.

Despite its strengths, DEVDAN is not without limitations. Its computational cost can be significant when dealing with high-dimensional data or large-scale streams, as the structural adjustments require frequent updates to the network parameters. Additionally, the choice of hyperparameters, such as those governing the NS thresholds, can influence the model's performance and may require careful tuning in specific applications.

In conclusion, DEVDAN represents a significant advancement in neural network design for evolving data streams. Its capacity to dynamically adapt its structure, leverage both labelled and unlabelled data, and address concept drift makes it a robust and flexible solution for real-world problems.

2.3.2.3 Adaptive Deep Learning (ADL)

The ADL framework is a self-organizing deep neural network designed for lifelong learning in dynamic and evolving environments, particularly when data streams exhibit concept drift (ASHFAHANI; PRATAMA, 2019). Unlike conventional Deep Neural Networks (DNNs), which rely on static architectures and predefined capacities, ADL features a flexible structure that evolves both in terms of network width (hidden nodes) and depth (hidden layers) to adapt dynamically to the nature of the incoming data. This adaptability ensures that ADL can efficiently handle non-stationary data distributions while mitigating issues such as catastrophic forgetting.

One of the standout features of ADL is its self-constructing network structure, which begins with no initial architecture and incrementally evolves as data streams are processed. The evolution of the network is driven by two core mechanisms: the NS formula and the Drift Detection Scenario (DDS). The NS formula governs the addition and pruning of hidden nodes within layers by evaluating the network's generalization power using bias and variance metrics. If high bias indicates underfitting, new nodes are added; conversely, high variance, signaling overfitting, triggers the removal of redundant nodes. This approach ensures that the network maintains an optimal balance between complexity and efficiency.

The DDS mechanism, on the other hand, manages the network's depth by detecting significant shifts in data distributions, known as concept drifts. When a drift is identified, a new hidden layer is introduced to capture the new data concept. Each layer in the network corresponds to a distinct time window of the data stream, effectively representing different concepts. This layer-specific representation is crucial for addressing catastrophic forgetting, as it enables the network to retain knowledge of old concepts while incorporating new ones.

Furthermore, ADL employs a complexity reduction strategy that merges redundant layers based on mutual information, ensuring that the network remains computationally efficient.

A key innovation of ADL is its different-depth structure, which connects each hidden layer directly to an output layer via a softmax function. This design allows each layer to produce a local output, which is then aggregated using a dynamic voting scheme to make the final classification decision. The voting weights are adjusted dynamically based on the relevance of each layer, enabling the network to emphasize layers that are more pertinent to the current concept. This architecture not only enhances the network's representational power but also provides a robust solution to catastrophic forgetting by preserving the relevance of older layers while adapting to new data.

ADL operates under the prequential test-then-train protocol, which reflects real-world scenarios where data streams arrive without labels. In this setup, incoming data is first used for testing and then for training. This single-pass learning approach ensures that the model remains efficient and scalable, even in environments with high data velocity or limited computational resources. To further enhance its learning capabilities, ADL incorporates dynamic resource allocation, which prioritizes the most relevant layers and nodes for parameter updates, reducing the risk of overfitting and ensuring stable performance over time.

Despite its strengths, ADL has some limitations. The model's reliance on statistical thresholds, such as those used in the NS and DDS mechanisms, requires careful calibration to achieve optimal performance across different datasets. Additionally, the computational overhead of managing a dynamically evolving network can be significant in scenarios with high data dimensionality or frequency. However, these challenges are mitigated by the framework's inherent flexibility and robustness, making it a powerful tool for real-world applications like fraud detection, adaptive recommendation systems, and real-time monitoring.

In conclusion, ADL represents a significant advancement in neural network design for evolving data streams. Its ability to construct and adapt its architecture autonomously, combined with mechanisms to address concept drift and catastrophic forgetting, sets it apart as a state-of-the-art solution for lifelong learning.

3 Just-in-Time Software Defect Prediction (JIT-SDP)

SDP is a critical field in software engineering, aiming to identify potential defects before they cause significant issues. JIT-SDP is a specialized approach that focuses on predicting defects at the moment a code change (commit) is made, offering a more proactive and precise alternative to traditional defect prediction methods. By leveraging historical commit data and machine learning techniques, JIT-SDP helps developers detect and mitigate potential defects early in the development process. To achieve accurate predictions, various models are employed in JIT-SDP, ranging from classical machine learning algorithms to more sophisticated online learning techniques capable of adapting to evolving software development patterns. This section introduces JIT-SDP, its fundamental concepts, and the models used to enhance its predictive capabilities.

3.1 What is JIT-SDP

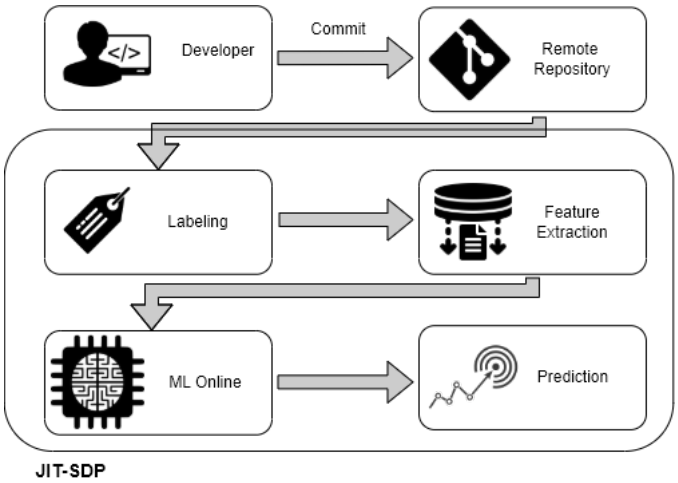
JIT-SDP is an approach in the field of software defect prediction. Unlike conventional methods that analyze entire modules or files, JIT-SDP focuses on predicting whether a specific code change will introduce defects soon after it is made. This granular approach has the main benefit of reducing the effort in code reviews and fixes, allowing developers to receive feedback immediately while the changes are still fresh in their minds, as described by (KAMEI; SHIHAB, 2016).

In agile development environments, where speed and quality are critical, JIT-SDP emerges as a strategic tool to reduce review costs and efforts, since defects detected early are cheaper and faster to fix. Studies such as that of (KAMEI et al., 2012) demonstrate that defective commits are often associated with specific characteristics, such as extensive changes to legacy code, high dispersion of modifications across files, or a history of frequent changes to critical subsystems, reinforcing the feasibility of predictive models based on commit metrics.

Wiht this, ŚLIWERSKI; ZIMMERMANN; ZELLER explored the relationship between defect tracking systems and version control systems to identify "fix-inducing changes,"meaning modifications in the code associated with previous defect fixes. They analyzed which characteristics of these changes were correlated with the introduction of new defects and observed that larger changes were more likely to induce future fixes. Expanding on this line of research, EYOLFSON; TAN; LAM demonstrated that factors such as the time of day, the day of the week, and the developer's daily commit frequency can influence the likelihood of a commit introducing defects.

Figure 3 provides an overview of the overall JIT-SDP workflow. The process begins when a developer commits to a remote repository, such as GitHub or GitLab. Each commit records changes to the source code, usually accompanied by a descriptive message that provides information about the purpose of the change. Once a commit is made, the system identifies the changes using APIs or webhooks. For example, in a GitHub integration, the webhook can be configured to trigger the system whenever a new commit is made, ensuring that recent changes are automatically detected in real time.

Figure 3 – Fluxogram of the JIT-SDP



Source: Author

Additionally, a labeler plays a key role in the process, especially in automated workflows. The labeler is not a predictive model, but a mechanism used to retrospectively assign labels to commits by correlating them with bug records in tracking systems such as Jira or GitHub Issues. This labeling process is crucial for generating labeled datasets that serve as ground truth for training machine learning models. The labeler can be configured to work automatically with tools like SZZ, a widely used algorithm for identifying defect-inducing commits in software repositories. Originally proposed by ŚLIWERSKI; ZIMMERMANN; ZELLER, SZZ works by linking bug reports to version control systems, allowing researchers and practitioners to trace which code changes were responsible for later fixes. The process consists of three main steps: first, it identifies commits that fix known issues by analyzing commit messages referencing bug reports in tracking systems; then, it traces back which previous commits introduced the lines of code that were modified in the bug-fixing commit; finally, these traced commits are flagged as defect-inducing, indicating they likely introduced the bug that was later fixed. There are several variants of SZZ such as RA-SZZ where a manual analysis of 1,000 changes labeled by RA-SZZ confirmed a 98.1% precision in refactoring detection (FAN et al., 2016). Futhermore, in a study involving 126,526 code changes across ten Apache projects, RA-SZZ produced the lowest rates of false positives and false negatives, making it the benchmark for evaluating defect prediction models. The high accuracy of SZZ (like RA-SZZ) in detecting refactorings is essential as it

minimizes false positives and negatives, directly improving the quality of the datasets used to train defect prediction models, which increases the reliability of interferences and reduces the costs associated with corrective maintenance.

However, the labeler itself does not predict whether a new commit will introduce defects; it only classifies past commits once sufficient information is available. In contrast, machine learning models aim to predict defect-inducing commits in real-time, before they reach later stages of development, enabling proactive defect prevention rather than retrospective analysis.

After identifying the changes, the features related to the modified commits are extracted. This is a crucial step for capturing relevant information that serves as input for prediction models. The feature extraction process involves analyzing various commit attributes, such as the number of modified files, the total lines of code added or removed, the time elapsed since the last modification of the affected files, the number of distinct contributors who previously modified those files, the entropy of the changes, among others. These features help quantify the scope and complexity of each commit, enabling machine learning models to distinguish between potentially defective and non-defective changes.

With the extracted features, the system uses machine learning models to predict the probability that the change will introduce a defect. These models are previously trained with historical data to identify patterns associated with failure-prone changes. For example, if extensive changes to legacy code are often associated with defects, the model can identify a 100-line change in an old file as a high-risk case. The result of this analysis is then sent to the developer.

3.2 Selected JIT-SDP work

Developers typically record all software changes as commits to a Source Code Management (SCM) system or a Version Control System (VCS). Software changes are typically smaller and each has a descriptive log message (KAMEI; SHIHAB, 2016). Therefore, when it comes to carry out code review for changesets predicted as defect inducing, it also likely requires less effort from the developers. In addition, since each change has a single committer, it is usually easier to track responsibilities and to conduct specialized code review activities for the change. Finally, developers may quickly examine their own potentially defect-inducing software changes as they make commits to the SCM, when a JIT-SDP model prompts them to do so.

Some works have demonstrated that certain implementations of JIT-SDP, such as LR-JIT, achieve higher predictive performance compared to other approaches. LR-JIT, introduced by (KAMEI et al., 2012), is a simple yet effective model that integrates 14 commit-level hand-crafted features with logistic regression to predict whether a commit is defective. These 14 features describe the properties of code changes, such as the size, diffusion, history, and author experience of code changes.

DeepJIT leverages Convolutional Neural Networks for text (textCNNs) to process both commit messages and code changes, automatically extracting meaningful semantic and syntactic features that contribute to defect prediction (HOANG et al., 2019). Unlike traditional JIT-SDP models that rely on manually engineered features, DeepJIT employs an end-to-end deep learning approach to learn representations directly from raw commit data. The model separately encodes commit messages using natural language processing techniques and represents code changes at a more granular level by analyzing added and deleted lines across multiple files. These extracted features are then combined to generate a comprehensive representation of the commit, which is used to predict whether a given change introduces defects.

However, JIT-SDP faces complex challenges in real-world scenarios. One of the most critical is the concept of drift. For example, development practices may evolve, new architectural patterns may be adopted. For instance, many organizations have transitioned from monolithic architectures to microservices (LAURETIS, 2019). In monolithic systems, a defect in one module may have predictable interactions with the rest of the system. In contrast, microservices introduce network dependencies and asynchronous communication, which can lead to novel failure modes, making traditional defect prediction models less effective. Another factor contributing to drift is team composition changes. As team members rotate, the experience profile of developers shifts, affecting code quality and review dynamics. Junior developers, for example, may introduce different types of defects compared to senior engineers. These changes cause static models, such as LR-JIT or DeepJIT to progressively lose accuracy, generating false positives (incorrect alerts) or false negatives (undetected defects).

MCINTOSH; KAMEI conducted a longitudinal analysis on the QT[1] and OPENSTACK[2] systems, encompassing 37,524 software changes from 2011 to 2014. Their findings emphasized that the characteristics of defect-inducing changes evolve significantly over time, a phenomenon that reduces the predictive performance of models trained on older data. They observed notable drops in the discriminatory power and calibration of models after a year, suggesting the need for frequent retraining with recent data to maintain effectiveness. This underscores the dynamic nature of software changes and the challenges it presents for maintaining accurate predictions over time.

CABRAL et al. expanded this research line by analyzing 10 open source projects hosted on GitHub, focusing on how the proportion of defect-inducing and clean examples changes temporally in JIT-SDP datasets. They identified class imbalance evolution as a significant factor, showing that tailored approaches capable of addressing these temporal fluctuations yielded superior predictive results. Their work highlighted the importance of adapting models to evolving data distributions to improve performance.

With this, strategies to address key challenges in this domain, such as adapting to

[1] <http://qt-project.org/>
[2] <http://www.openstack.org/>

class imbalance dynamics and managing verification delays were proposed (CABRAL et al., 2019). The authors demonstrated that class imbalance in JIT-SDP evolves over time, impacting predictive performance by altering the distribution of defect-inducing and clean software changes. Traditional approaches assume a static imbalance ratio, which leads to suboptimal predictions as the imbalance fluctuates. Despite these advances, JIT-SDP continues to face difficulties, particularly in handling noisy data, where error amplification remains a significant concern. These challenges underscore the need for more resilient methods capable of adapting to the dynamic and evolving nature of data streams.

JIT-SDP in scenarios where classifiers are updated incrementally as new training data arrives is investigated by (TAN et al., 2015). While previous studies considered verification latency in online defect prediction, the authors evaluated seven updatable algorithms, including IBK and KStar (instance-based methods), NNge (nearest-neighbors), Bayes and LWL (Naive Bayes), LogitBoost (boosting), and SPegasos (Support Vector Machines). To tackle class imbalance, they applied resampling techniques. Cross-Project (CP) JIT-SDP in realistic online learning settings was studied by (TABASSUM et al., 2020). In this context, CP JIT-SDP refers to training a classifier using historical defect data from other projects, which helps when a new project lacks sufficient labeled training data. Conversely, Within-Project (WP) JIT-SDP relies exclusively on defect data from the same project to train the classifier, updating it incrementally as new changes occur. The authors found that combining CP and WP data enhances predictive performance, particularly in early project stages or during concept drift. Their results showed that a unified classifier using both CP and WP data outperforms WP-only approaches. However, they noted challenges, including the limited effectiveness of ensemble classifiers in online learning due to insufficient training data and the importance of ensuring CP and WP data similarity to address concept drift.

The analysis of JIT-SDP approaches reveals significant progress in the ability to predict defects soon after code changes, especially in scenarios with dynamic data. However, these works face recurring challenges, such as class imbalance, noisy labels, and concept drift. Although models such as LR-JIT and DeepJIT have contributed to the evolution of the field, they have limitations that make them unsuitable for certain practical situations. For example, the dependence on large-scale labeled data, low adaptability to continuous changes in data streams, and high computational complexity limit their applicability in online and non-stationary scenarios. In this context, our model stands out for its ability to integrate semi-supervised learning and adaptive strategies, reducing the need for labels and offering better performance in non-stationary data streams.

Table 1 presents a comparison between these models and our model, highlighting their main features, advantages, and limitations. The LR-JIT method, while simple and efficient for static data, struggles with concept drift. In contrast, the proposed method is designed to handle non-stationary data streams, making it a better fit for real-world scenarios where data

evolves over time. Deep-JIT stands out for its automatic feature extraction using CNNs, but its heavy reliance on large amounts of labeled data limits its practicality. The proposed method, by leveraging semi-supervised learning, reduces this dependency and remains effective even with fewer labeled samples. The proposed method offers a more lightweight and adaptable solution.

CP+WP JIT-DP improves defect prediction in early-stage projects by combining intra- and inter-project data, but it struggles to maintain performance under intense concept drift. The proposed method, however, is better suited to adapt to continuous changes in data distribution. Finally, incremental algorithms allow continuous updates, but without specific drift adaptation strategies, their effectiveness is limited. The proposed method integrates these adaptation techniques while also reducing reliance on labeled data, making it a more robust choice for dynamic learning environments.

Table 1 – Comparision of methods

Work	Approach	Strengths	Limitations
LR-JIT (KAMEI et al., 2016)	Logistic regression with 14 features	Simplicity and efficiency in static data	Low adaptation to concept drift
Deep-JIT (HOANG et al., 2019)	CNN applied to commit text	Automatic feature extraction	Reliance on large volumes of labeled data
CP+WP JIT-SDP (TABASSUM et al., 2020)	Intra- and inter-project data combination	Improved performance in early project scenarios	Limited effectiveness in intense concept drift
Incremental Algorithms (TAN et al., 2015)	Various incremental algorithms	Incremental upgrade capability	Reduced effectiveness without specific strategies for concept drift height

Source: Author

These related works were not used directly in the experiments since they do not meet the specific requirements of the studied scenario, which demands high adaptability and low dependence on labeled data. S3WA, in turn, was designed to address these limitations, offering a more robust and efficient solution for continuous learning environments.

4 S3WA

Section 4 of this master thesis introduces the neural network S3WA model, an innovative approach that integrates semi-supervised learning techniques with SWA to handle non-stationary data streams, with a focus on JIT-SDP scenario. The primary objective of this section is to detail the architecture and learning algorithm of S3WA, demonstrating how it adapts to concept drift and maintains high predictive accuracy in dynamic environments. By combining the robustness of semi-supervised learning with the efficiency of weight averaging, S3WA aims to overcome the limitations of traditional models, offering a more adaptable and efficient solution for real-time defect prediction.

4.1 Network Architecture

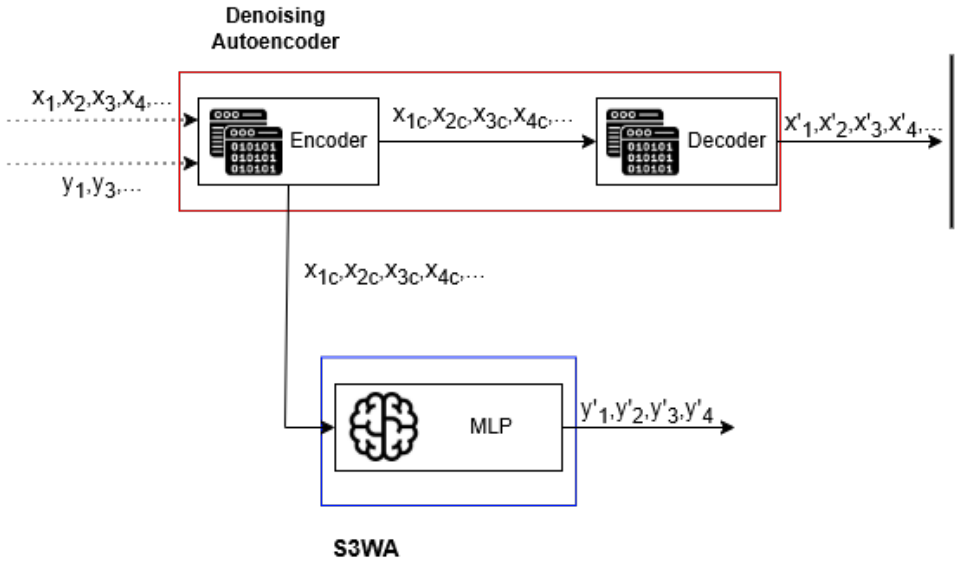
Figure 4 presents the S3WA model, a neural network method where employs a Denoising Autoencoder (DAE) (VINCENT et al., 2010) coupled with a fully connected softmax output layer. The DAE learns useful data representations from labeled and unlabeled data, while the output layer discriminates classes using encoded features from the encoder's output. Such an architecture allows the use of the algorithm in semi-supervised learning and provides a better initialization of hidden layers in deep architectures.

Figure 4a illustrates the testing process, where both labeled and unlabeled data are processed by the model to generate predictions. In contrast, Figure 4b represents the training process, where only labeled data is used to adjust the model parameters. The reason for this distinction is that, during training, labeled data provides a reference for refining the feature representations learned by the model, ensuring better generalization. During testing, the model applies the learned representations to both labeled and unlabeled data, enabling predictions even in cases where labels are unavailable.

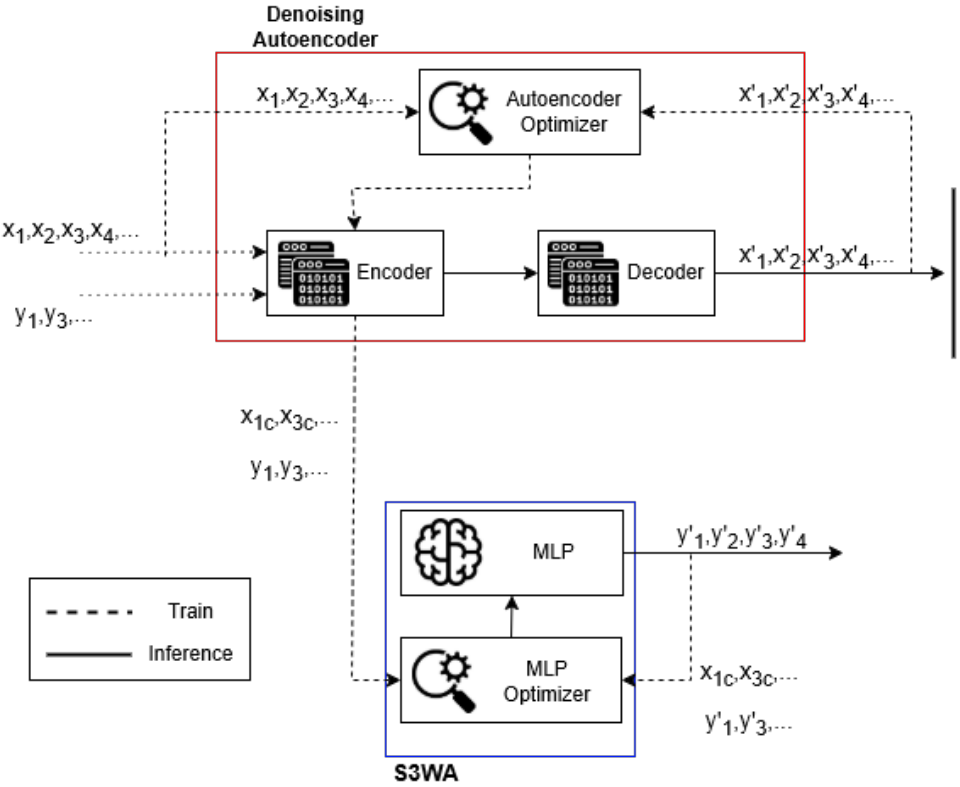
The DAE reconstructs an original input example x_t from the normal noise perturbation $\tilde{x}_t = x_t + pN(0,1)$, where p is the amount of added noise. In Figure 4b, S3WA uses the encoder's output $z_e = h_e(\tilde{x}_t W_t)$, where z_e corresponds to the encoded output of the corrupted input $\tilde{x}_t$. The decoding scheme of the DAE's reconstruction process is given by $z = h_d(z_e W_t')$, where W_t' are the decoder weights in the reconstruction phase, and h_e and h_d are the activation functions of the encoding phase and decoding phase, respectively (ASHFAHANI et al., 2020). Here, $W_t \in \mathbb{R}^{D \times H}$ is the weight matrix and H is the number of hidden nodes. Tied weights are used in the DAE, ensuring that W_t' is the transpose of W_t, i.e, $W_t' = W_t^\top$ (VINCENT et al., 2010).

The reconstruction error in the DAE is measured using the mean squared error (MSE):

$$\text{MSE} = \frac{1}{|B_t|} \sum_{x \in B_t} (x_t - z_t)^2, \tag{4.1}$$



(a) Test process



(b) Train process

Figure 4 – S3WA Architecture

where x_t and z represent the input and reconstructed output at each time step, respectively. This process of adding Gaussian noise and reconstructing the input forces the DAE to learn robust representations that can capture the underlying structure of both labeled and unlabeled data.

After being processed by the DAE, the encoder's output h_t is passed to a fully connected Multilayer Perceptron (MLP) for classification. The MLP performs the classification task by predicting the posterior class probabilities $f(W, x)$, with the loss function being the multiclass cross-entropy:

$$\text{Loss}_{\text{softmax}} = -\sum_i y_i \log(\hat{y}_i), \tag{4.2}$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability for class i. The softmax output layer, combined with the encoded features from the DAE, ensures the model can make accurate predictions in semi-supervised learning settings.

The S3WA framework processes the data in chunks B_t, utilizing a sliding window scheme. Each chunk $B_t = \{x_{s_t}, x_{s_t+1}, \ldots, x_{s_t+n_c-1}\}$, where $s_t = (t-1) \cdot s$, n_c is the chunk size, and s is the step size of the sliding window, represents a portion of the continuous data stream. When $s < n_c$, consecutive chunks overlap, allowing examples to be processed multiple times for enhanced robustness. During the test-then-train strategy, all data in the chunk is first tested with the model's current parameters, ensuring that the evaluation respects the temporal order of the data. After the testing phase, only labeled examples within the chunk are used to train the model. The DAE uses labeled inputs to adjust the encoder and decoder weights, while the MLP updates its parameters using the cross-entropy loss from the labeled data.

The sliding window mechanism ensures that the model processes data incrementally, maintaining a balance between computational efficiency and the ability to adapt to gradual changes in the data stream. The chunk size n_c and step size s are critical parameters that influence the model's performance. Larger chunks provide more data for training and testing but increase latency, while smaller chunks may lead to insufficient information for effective learning. Similarly, smaller step sizes ($s \ll n_c$) result in higher overlap and redundancy, while larger step sizes ($s = n_c$) may cause the loss of temporal continuity.

After a predefined number of steps, determined by the parameter start, the SWA mechanism is activated to smooth the model weights and improve generalization. SWA ensures that the model captures general trends in the data rather than overfitting to short-term fluctuations, thereby enhancing stability and robustness.

The integration of the DAE, MLP, sliding window-based chunking, and SWA makes S3WA a powerful tool for semi-supervised learning in dynamic data streams. The DAE provides robust representations that are resilient to noise, while the MLP ensures accurate classification. The test-then-train strategy, combined with the sliding window mechanism, allows the model to adapt continuously to changes in the data stream while respecting its temporal order. This

architecture is particularly well-suited for applications such as just-in-time defect prediction and other real-time learning tasks, where the ability to process labeled and unlabeled data efficiently is crucial.

4.2 Learning Algorithm

For each layer in the DAE and the softmax output layer, S3WA has a weight vector $\mathbf{w}_t \in W_t$, trained with SGD; and maintains a weight vector $\boldsymbol{\mu}_t \in M_t$, trained with our novel online weight averaging algorithm. The weights $\mathbf{w}_t$ correspond to a neural network that is being trained on the current concept, whereas the averaged vector of weights $\boldsymbol{\mu}_t$ represents the version of an ensemble composed of previously trained weights $\mathbf{w}$ on the current concept. The weights $\boldsymbol{\mu}_t$ are used for prediction after time step θ. The weights $\mathbf{w}_t$ are updated continuously with new examples to reflect the current concept and are periodically incorporated into $\boldsymbol{\mu}$ through weight averaging. This enables $\boldsymbol{\mu}_t$ to become a better ensemble-like model through the addition of knowledge from models trained on the current concept. This mechanism exploits the fact that SGD finds solutions on the borders of wide minima so that the averaged weights become more robust and lead to better generalization (IZMAILOV et al., 2018).

The SWA is an effective approach for neural networks in offline batch learning due to its ability to find wide optima via weight averaging (IZMAILOV et al., 2018). These minima produce models that are more robust to noise. It delivers better predictive performance with a near negligible increase in computational time compared to other optimizers. The weights in SWA are averages across batches:

$$\boldsymbol{\mu}_{t+1} = \frac{n\boldsymbol{\mu}_t + \mathbf{w}_{t+1}}{n+1}, \tag{4.3}$$

where n is the number of averaged models. To derive this formulation to data stream learning, we can rewrite

$$\mathbf{w}_{t+1} = \boldsymbol{\mu}_t + (\mathbf{w}_{t+1} - \boldsymbol{\mu}_t),$$

then, replacing have the latter into the former, we have

$$\begin{aligned}\boldsymbol{\mu}_{t+1} &= \frac{n\boldsymbol{\mu}_t + \boldsymbol{\mu}_t + (\mathbf{w}_{t+1} - \boldsymbol{\mu}_t)}{n+1} \\ &= \frac{(n+1)\boldsymbol{\mu}_t + (\mathbf{w}_{t+1} - \boldsymbol{\mu}_t)}{n+1} = \boldsymbol{\mu}_t + \frac{\mathbf{w}_{t+1} - \boldsymbol{\mu}_t}{n+1}\end{aligned}$$

Let $\alpha = \frac{1}{n+1}$, our updated model no longer depends on the number of points (models) n and now can be used in online training with the proper tuning of α.

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t(\mathbf{w}_{t+1} - \boldsymbol{\mu}_t) \tag{4.4}$$

If we define $\alpha = 1 - \beta$, this formulation becomes $\boldsymbol{\mu}_{t+1} = \beta\boldsymbol{\mu}_t + (1-\beta)\mathbf{w}_{t+1}$, a Exponential Moving Average (EMA) with β as a fading factor. This shows that S3WA can be interpreted as an adaptive gradient optimization algorithm, in line with various state-of-the-art optimization methods (KINGMA; BA, 2014).

The hyperparameter α can be interpreted as a learning rate that regulates the influence of

new weights $\mathbf{w}_t$ in the averaged model $\boldsymbol{\mu}_t$. Smaller α is desired for stable periods, in which the knowledge obtained in previously trained models is important. Therefore, the current averaged model M becomes a robust ensemble of past relevant neural networks. With larger α, new weights trained with SGD have more importance in the current model $\boldsymbol{\mu}_t$, whilst older weights at previous time steps are gradually forgotten. This might be necessary for periods with concept drifts, in which new models should acquire important recent knowledge and older ones become obsolete and should be decayed in the averaged model. The proper tuning of α might lead to better generalization since the model could learn new and important information while being able to forget obsolete past knowledge. The hyperparameter θ controls when S3WA starts the training of $\boldsymbol{\mu}$ and producing predictions. It controls the amount of training $\mathbf{w}_t$ should receive before starting to be incorporated into $\boldsymbol{\mu}_t$. This can be interpreted as a warm start for S3WA.

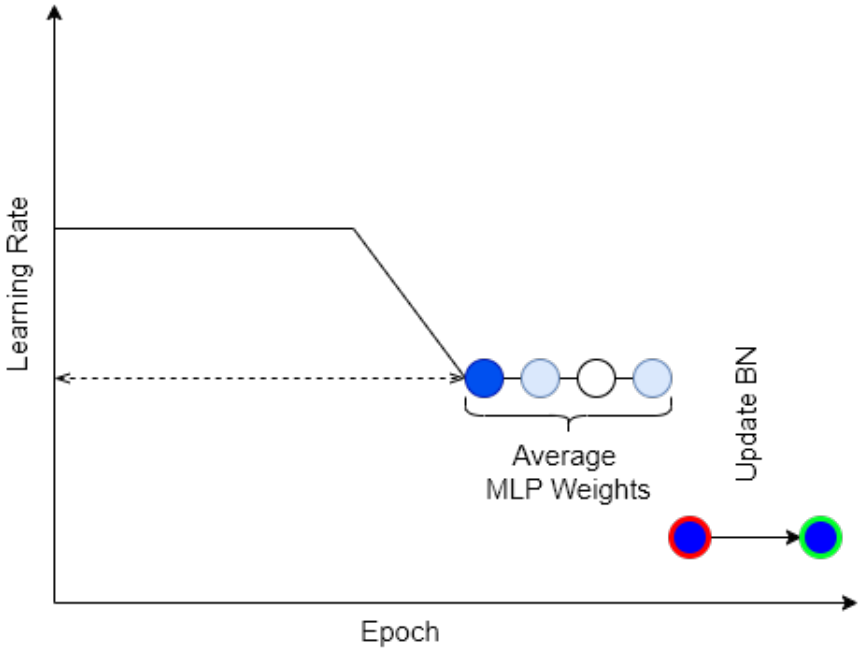
The S3WA model combines the efficiency of SWA with the online learning capability of Stochastic Gradient Descent SGD to handle continuous data streams. Initially, training begins with pure SGD, where the weights $\mathbf{w}_t$ are iteratively adjusted based on the available samples, allowing the network to learn patterns without applying weight averaging. After a certain period, the transition to SWA begins, where the weight averaging technique is applied. At this stage, the average weight vector $\boldsymbol{\mu}_t$ is progressively updated as new weights $\mathbf{w}_t$ are learned, following the equation 4.4 which reduces weight variability and helps in converging to more generalizable solutions.

To implement this approach, a specific learning rate schedule is adopted, as illustrated in the example in Figure 5. During the first 75% of training, a standard decaying schedule is applied to allow the model to explore different regions of the weight space efficiently. In the final 25%, a high constant learning rate is used, and weight averaging is performed over multiple iterations. This transition ensures that the model benefits from both the exploration provided by SGD and the stability of SWA. At the end of the training process, batch normalization (BN) statistics are updated to reflect the final averaged model parameters.

With this approach, the model balances the exploration of new regions in the weight space, a characteristic of SGD, with SWA's ability to find more stable and better generalizing solutions.

Alg. 1 shows the S3WA algorithm. It continuously trains a neural network $f(W, \mathbf{x})$ with SGD, which is responsible for S3WA's predictions until time step θ. After θ, the predictions are produced by the averaged model M_t. If instance $\mathbf{x}_t$ is labeled, then all layers are trained. Otherwise, only the DAE weights are updated. The averaged weight vector $\boldsymbol{m}u$ are trained following Eq. 4.4.

Figure 5 – Illustration of the learning rate schedule adopted by SWA.



Source: (IZMAILOV et al., 2018) (Adapted)

Algorithm 1 S3WA algorithm.

Input: Stream $\mathcal{S}$, α, θ, $\mathbf{w} \in \mathbb{R}^k$
Output: Trained neural network and predictions on $\mathcal{S}$

while $s_t = (x_t, y_t) \neq \emptyset$ **do** ▷ $t \leq$ end of stream
  $B_t \leftarrow \{s_{t-N+1}, \ldots, s_{t-1}, s_t\}$
  if $t \leq \theta$ **then** ▷ S3WA's learning starts at time step θ
    $\hat{y} \leftarrow \text{argmax}_i(f_i(W, x_t))$ ▷ Predictions are made with w
    Update each w using SGD
  else
    $\hat{y} \leftarrow \text{argmax}_i(f_i(M, x_t))$ ▷ Prediction is made by S3WA
    Update each w_t using SGD
    if y_t is defined **then** ▷ Instance x_t is labeled
      Update output layer with (x_t, y_t) using SGD
    end if
    Update averaged weights $\boldsymbol{\mu}$ as in Eq. 4.4 ▷ Incorporating new weights w to μ
  end if
  $t \leftarrow t + 1$ ▷ Next time step
end while

4.3 S3WA's optimization paths

The understanding of the geometric properties of surfaces produced by loss functions has been proved to be promising in the neural networks optimization field. (KESKAR et al., 2016) and (CHAUDHARI et al., 2019) conjuncture that the width of local optimum is directly related to model's generalization capacity showed, for example, in test performance. The key concept

is that wider optima, i. e converged solution points of a given procedure, produce better model stability and robustness. The therms wide, broad and large solutions have similar meanings and are commonly used to classify regions in weight space containing points whose performance tend to be alike. This property gives an advantage to broad regions of high-performing networks and builds its importance. (IZMAILOV et al., 2018) explains these aspects graphically illustrating that there exists a shift between the train loss and test error surfaces produced by weights (even if they are qualitatively comparable), suggesting that points centralized in larger optimal zones are more robust under small perturbations and can lead to better generalization. Later, such points are argued to be reached by simply averaging weights that fit the boundaries of these regions. SWA is an example of optimization algorithm able to find wide optimal areas in weight space, that applies this strategy and improves generalization in stationary classification scenarios,

Despite its features, SWA was proved to not maintain its performance in data stream learning environment. The explanation is that since the start of the stream, SWA keeps every averaged weights in track, and when concept drifts occurs, the weights produced by past concepts are kept into the count. In fact, after a concept drift, SWA is no longer capable of reach wide optimal regions as before the drift. In contrast, S3WA not only is capable of finding broad optima but this property holds for data stream learning even in the presence of concept drifts.

To illustrate this fact, a *Loss Function Visualization Method* was used to show the weight space around the solutions proposed by S3WA and SWA during the streaming process. Such methods applies dimensionality reduction techniques to visualize an objective function (e. g. loss or test error) values within the parameter space of the model. Izmailov et al. in (IZMAILOV et al., 2018) used a visualization method based on projection presented in (GARIPOV et al., 2018) to plot the training loss and test error to show SWA's ability to find wide and broad optima.

The method presented by Izmailov et al. (GARIPOV et al., 2018) considers the existence of three initial weight vectors $\mathbf{w_1}$, $\mathbf{w_2}$, and $\mathbf{w_3}$, through which the vectors $\mathbf{u}$ and $\mathbf{v}$ can be obtained, where

$$\mathbf{u} = \mathbf{w_2} - \mathbf{w_1}$$

$$\mathbf{v} = (\mathbf{w_3} - \mathbf{w_1}) - \frac{\langle \mathbf{w_3} - \mathbf{w_1}, \mathbf{w_2} - \mathbf{w_1} \rangle}{||\mathbf{w_2} - \mathbf{w_1}||^2} \cdot (\mathbf{w_2} - \mathbf{w_1})$$

Then, the normalized vectors $\hat{\mathbf{u}} = \mathbf{u}/||\mathbf{u}||$ and $\hat{\mathbf{v}} = \mathbf{v}/||\mathbf{v}||$ form an orthogonal basis in the plane containing $\mathbf{w_1}$, $\mathbf{w_2}$, and $\mathbf{w_3}$. To visualize the error surface in this plane, a Cartesian grid is defined in the $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$ basis, and the neural network is evaluated at each point of the grid. A weight vector $\mathbf{P}$, with coordinates (x, y) in the plane, is given by:

$$\mathbf{P} = \mathbf{w_1} + x \cdot \hat{\mathbf{u}} + y \cdot \hat{\mathbf{v}} \tag{4.5}$$

Finally, to obtain the coordinates $(\tilde{x}, \tilde{y})$ given a weight vector $\mathbf{P}$, we do:

$$\tilde{x} = (\mathbf{P} - \mathbf{w_1}) \cdot \hat{\mathbf{u}} \tag{4.6}$$

$$\tilde{y} = (\mathbf{P} - \mathbf{w_1}) \cdot \hat{\mathbf{v}} \tag{4.7}$$

Two different models using SWA and the S3WA averaging methods were tested in Sine1 data stream. For a fair comparison, both the models uses S3WA's proposed architecture and hyper parameters, i. e. they only differs in the averaging equation. We the depict an abrupt concept drift, which length is one time step, that occurs in time step 4000. We then use 3 different parameter vector in the optimization trajectory as the subset $\mathbf{w_1}$, $\mathbf{w_2}$, $\mathbf{w_3}$ to form the cartesian basis ($\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$) to define a 2-dimensional plane in the weight space containing all affine combinations of these points.

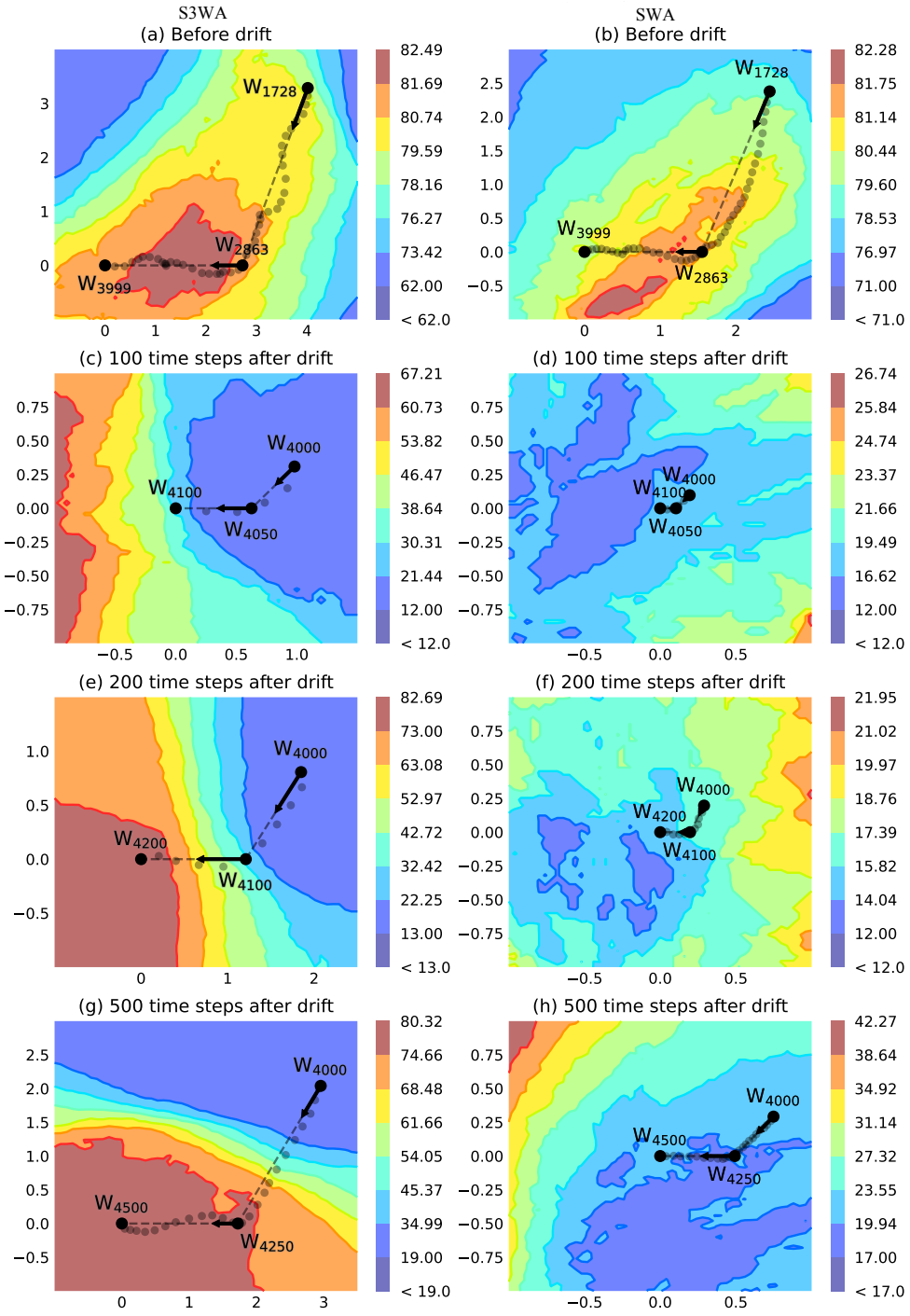
Figure 6 illustrates the optimization paths of the S3WA and SWA models during an abrupt concept drift in the Sine1 data stream, this dataset has two relevant attributes, each attributes has values uniformly distributed in [0, 1]. In the first context all points below the curve y = sin(x) are classified as positive. After the context change the classification is reversed (GAMA et al., 2004).

The figure 6 consists of plots showing the prequential accuracy surface, representing the evolution of model accuracy over time. The dots on the graphs indicate the solutions found by the models, while the arrows show the direction of optimization. The concept drift occurs at time step 4000, and the figure allows us to observe how the models behave before and after this event.

Before the drift, both S3WA and SWA converge to stable solutions, with prequential accuracy increasing over time. In Figure 6a, the S3WA shows a smoother optimization trajectory, indicating that the model is finding robust and generalizable solutions. In offline learning environments, SGD finds solutions on the edges of wide minima, and averaging these solutions produces an even wider minimum (IZMAILOV et al., 2018). Figure 6 demonstrates that, with S3WA, this property holds even in data stream learning, even in the presence of concept drifts. During a stable period (before the drift), S3WA is able to find a path through a wide local minimum, as shown in Figure 6a, demonstrating the usefulness of weight averaging in data stream learning. SWA, on the other hand, also shows stable convergence but with a slightly more erratic trajectory, as seen in Figure 6b, suggesting that it may be more susceptible to fluctuations in the data. This occurs because SWA, although effective in finding wide solutions, lacks the same dynamic adaptation mechanism as S3WA, which combines weight averaging with EMA to handle changes in the data.

After the drift, which occurs at time step 4000, S3WA demonstrates more efficient adaptation, quickly adjusting to the new concept and maintaining a stable optimization trajectory. This is evidenced by the smoothness of the curves and the consistent direction of the arrows in Figures 56c and 6e, indicating an effective search for optimal solutions. When an abrupt drift occurs, the optimization surface changes, but S3WA is able to recover within a few time steps due to the use of EMA for weights, which allows forgetting old concepts. The method was able to correctly learn the new concept and forget the previous one, demonstrating the adaptability potential of EMA for weights. On the other hand, SWA struggles to adapt to abrupt changes. Its optimization trajectory becomes more irregular, with sharp changes in the direction of optimization, as seen in Figures 6d and 5e, indicating lower robustness in handling concept

Figure 6 – Plots of the prequential accuracy surface depicting S3WA and SWA optimization paths during an abrupt concept drift on Sine1 data stream.



Source: Author

drifts. Unlike S3WA, SWA relies solely on weight averaging to maintain stability, which is insufficient to quickly adjust to new data distributions.

In Figure 6e, S3WA continues to stabilize its optimization path, finding a new wide minimum that corresponds to the post-drift data distribution. By Figure 6g, S3WA has fully adapted to the new concept, maintaining a smooth and consistent trajectory. In contrast, SWA, as shown in Figures 6ef and 6h, continues to exhibit erratic behavior, with frequent and abrupt changes in its optimization direction, highlighting its inability to fully recover from the drift.

5 RESULTS

JIT-SDP is a crucial field for ensuring software quality during the development process. In dynamic development environments, such as code repositories, the frequent introduction of new features and modifications imposes significant challenges for machine learning models, as the underlying data patterns are constantly evolving. As a result, one of the main obstacles faced by traditional defect prediction models is the change in data patterns over time, known as concept drift. Sections 5.1 and 5.2 precisely explore these challenges, analyzing the performance of machine learning models in non-stationary environments and their specific application in the JIT-SDP context.

Section 5.1 establishes a comparative basis with previous studies, allowing the evaluation of the proposed models' performance in different concept drift scenarios. For this purpose, artificial and real-world datasets widely adopted in the state of the art are used, simulating different types of changes in data patterns over time. The goal of this section is not to specifically evaluate software defect prediction but rather to analyze how machine learning models handle non-stationary environments in general. The experiments in this section include scenarios with abrupt and gradual drifts, enabling an understanding of which online learning techniques are more effective in adapting to these variations. Additionally, different learning approaches, such as batch learning online and chunk-based learning, are tested to determine which one is more suitable for handling dynamic data streams.

Section 5.2, on the other hand, represents the central focus of the master thesis, as it applies the previously analyzed techniques to the specific problem of JIT-SDP. In this part, the experiments are conducted using JIT-SDP specific datasets, which contain information about source code changes and the occurrence of defects over time. This approach allows for assessing the effectiveness of the models in a real software development context, where it is crucial to predict defects at the moment commits are made. The main objective of this section is to compare the performance of the proposed model S3WA with other online learning approaches, verifying which one is more efficient in detecting defects in real time. Additionally, this section investigates the impact of semi-supervised learning in reducing the dependency on labeled data, a critical aspect to making models more practical and applicable in real-world scenarios.

The primary goal of this dissertation is to develop a robust semi-supervised machine learning model for JIT-SDP, capable of handling dynamic and unpredictable environments. The research conducted in Sections 5.1 and 5.2 is essential to this objective, as it enables the comparison of S3WA's efficiency with other existing approaches, identifying their advantages and limitations in adapting to concept drift scenarios and practical JIT-SDP applications.

5.1 Results with state-of-the-art datasets

5.1.1 Experimental Method

In Table 2 summarized the artificials and reals streams. Artificial data streams are generated as sequences of different functions $\{r_i\}$, where each function represents a specific concept within the data generator. These functions define distinct patterns in the data, which may change over time. To illustrate this concept, consider a climate simulation: one function r_1 might represent a period of stable weather, while another function r_2 could represent a phase with sudden temperature fluctuations. The purpose of these functions is to simulate scenarios where the environment undergoes transformations, reflecting real-world events such as seasonal changes or variations in financial and demographic patterns. The drifts are abrupt and gradual, taking 4000 time steps for the gradual drift and one time step for the abrupt drift. This data streams include 5% of labeled data using binary mask.

In order to assess the classifiers' capacity to adjust to growing concept drift, we employed four different data stream generators that produced streams with a range of concept drift types. The SEA generator (STREET; KIM, 2001) is a artificial data stream generator that simulates a binary classification problem by defining a linear decision boundary in a multidimensional numerical space, making it equivalent to a simple hyperplane separation problem. The SEA consists of a binary classification problem in a three-dimensional space, where only two variables are relevant while the third acts as noise. The class separation is determined by a linear decision boundary, which can be altered over time to simulate abrupt or gradual changes in data patterns. To predict the groupings of individuals, the Agrawal generator (AGRAWAL; IMIELINSKI; SWAMI, 1993) was designed to generate structured records that simulate realistic scenarios of demographic and financial data. The dataset consists of nine attributes, including age, salary, education level, and credit history, generated through deterministic rules that define the relationship between these attributes and the target class.The generator produces records classified into two distinct groups (Group A and Group B), based on rules defined from combinations of numerical and categorical attributes. Sine generator was explain in Section 4.3. A category attribute called STAGGER (SCHLIMMER; GRANGER, 1986) has the ability to forecast the binary function of object lists. It consists of a binary classification problem where instances are described by three categorical attributes: size (small, medium, large), color (red, blue, green), and shape (circle, square, triangle).

We also evaluated classifiers using real-world data streams. The data stream Electricity was collected from the Australian New South Wales Electricity Market (HARRIES; WALES et al., 1999). This dataset has seasonality of prices with sensitivity to short-term events such as weather fluctuations. Each example of the dataset has five attributes: the day of the week, the time stamp, the NSW electricity demand, the Vic electricity demand and the scheduled electricity transfer between states. Class label represents the change of the price related to the

Table 2 – Summary of data streams.

Stream	Concept sequences	Number of inst.	Dim.
Artificial data streams			
Sine1	$r_1 \rightarrow r_2 \rightarrow r_1$	12000	4
Sine2	$r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4 \rightarrow r_1$	20000	4
Agrawal1	$r_1 \rightarrow r_3 \rightarrow r_7 \rightarrow r_{10}$	20000	36
Agrawal2	$r_1 \rightarrow r_4 \rightarrow r_6 \rightarrow r_5 \rightarrow r_2 \rightarrow r_9$	24000	36
Agrawal3	$r_1 \rightarrow r_2 \rightarrow r_1 \rightarrow r_3 \rightarrow r_4$	20000	36
Agrawal4	$r_1 \rightarrow r_3 \rightarrow r_6 \rightarrow r_7 \rightarrow r_5 \rightarrow r_4$	20000	36
SEA1	$r_1 \rightarrow r_3 \rightarrow r_2$	20000	3
SEA2	$r_1 \rightarrow r_3 \rightarrow r_1 \rightarrow r_3 \rightarrow r_2$	20000	3
STAGGER1	$r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_2$	16000	7
STAGGER2	$r_1 \rightarrow r_3 \rightarrow r_2 \rightarrow r_4$	16000	7
Real-world data streams			
Electricity	-	27549	7
NOAA	-	18159	8
Power Supply	-	29928	2
Sensor	-	130073	5
Chess	-	503	8
Luxembourg	-	1901	31

last 24 hours. With the categorization job of predicting whether it rained on a given day, the NOAA stream (NOAA, 2012) has several decades' worth of daily observations of temperature, pressure, visibility, wind speed, and other meteorological data. Finding the hour of the day that a particular power supply measurement was recorded at is the responsibility of the Power Supply stream (ZHU, 2010), which contains three years' worth of hourly measurements of the energy supply. The classifier's goal is to determine which sensor produced a particular measurement from the Sensor stream (ZHU, 2010), which is made up of temperature, humidity, light, and sensor voltage values gathered from sensors in a research facility. To this paper we used only two largest classes in this stream: sensors 29 and 31.

Chess game records with seven attributes from December 2007 to March 2010 are part of the Chess data stream (ŽLIOBAITĖ, 2011). Every participant has a rating that fluctuates over time. Concept drift can be caused by players' participation in different competitions and tournaments as well as skill changes over time. The European Social Survey from 2002 to 2007 serves as the foundation for the Luxembourg data stream (ŽLIOBAITĖ, 2011). A total of 1,901 instances were gathered during a five-year period, each of which represents a subject whose characteristics are based on survey replies. Predicting whether a subject uses the internet heavily or little is the classification task because internet usage varies over time and can lead to concept drifts. These real-world streams, also have three versions with 5% of labeled data, totaling 6 real-world streams.

We are making a comparision between S3WA and seven online learning approaches that use neural network as their base: DEVDAN, ADL, Parsnet, Ozabag-Adwin, DWM, CSB2

and SRPC. However, DEVDAN, ADL and Parsnet uses in yours processing method the batch processing and semisupervised learning, while the models Ozabag-Adwin, DWM, CSB2 and SRPC utilizes chunk processing with sliding window and supervised learning. Classifiers get access to a data stream only once, and they are trained and assessed on that stream at that one chance in order to imitate real-world learning scenarios in this and the subsequent experimental setups. Every approach performs prequential training and assessment (GAMA; SEBASTIAO; RODRIGUES, 2013), meaning that classifiers are assessed and then trained with each incoming instance.

We separate the initial 10% of each data chosen data stream to create a validation stream to perform the hyperparameter tuning of each model: Ozabag-Adwin, CSB2, SRPC and DWM. We realize a random search with 1000 iterations on the created validation streams following the probability distributions in Table 3 (BERGSTRA; BENGIO, 2012). For ADL, DEVDAN and Parsnet, we use the same parameters used in your respective articles.

The perfomances of chosen models were evaluated on the streams with prequential accuracy estimator with a fading factor of 0.999 (GAMA; SEBASTIAO; RODRIGUES, 2013).

Prequential accuracy is a metric used in online learning to continuously assess a model's performance (GAMA; SEBASTIAO; RODRIGUES, 2013). Instead of separating training and testing sets, the model makes a prediction for each new instance, records the accuracy, and then updates its parameters using that instance. This method reflects the model's performance over time and allows for concept drift detection, as it shows how accuracy varies as new data is introduced. A forgetting factor (α) can be applied to give more weight to recent instances, making the metric more sensitive to changes in data patterns.

In other words, if a model is trained to predict software defects from commit streams, prequential accuracy enables continuous performance evaluation, capturing the impact of code changes over time. A drop in accuracy may indicate concept drift, meaning the relationship between input data and labels has changed.

Prequential accuracy with a forgetting factor α is defined as:

$$A_t^\alpha = \frac{\sum_{i=1}^{t} \alpha^{t-i} 1(\hat{y}_i = y_i)}{\sum_{i=1}^{t} \alpha^{t-i}} \tag{5.1}$$

Where:

$\alpha \in (0, 1]$: forgetting factor that determines the decay rate of weights. Values closer to 1 indicate slower forgetting;

$\hat{y}_i$: model's prediction for example i.

y_i: true label for example i.

$1(\hat{y}_i = y_i)$: indicator function, which equals 1 if $\hat{y}_i = y_i$ (correct prediction) and 0 otherwise.

The denominator $\sum_{i=1}^{t} \alpha^{t-i}$ normalizes the weights, ensuring that the accuracy remains in the range $[0, 1]$.

The forgetting factor α controls the weight assigned to past observations: For $\alpha = 1$: all examples are weighted equally, equivalent to standard prequential accuracy; For $\alpha < 1$: recent examples receive higher weights, and older examples are exponentially down-weighted.

To ensure a fair evaluation of computational cost across different models, we analyzed their execution times and processing methods, the processing time was measured from the first instance of the dataset to the last, using the time function from Python's standard library. The total execution time for each method was recorded by capturing the initial timestamp before processing the first instance and the final timestamp after processing the last instance, ensuring an accurate assessment of the computational performance of each approach.

This analysis considers not only the total processing time, but also the scalability of the models in the face of dynamic data flows. After evaluation, each commit was integrated into the training set, allowing the models to update their parameters and adapt to new information. This process replicated real-world conditions of continuous data streams, simulating the online arrival of commits in software repositories.

5.1.2 Experiments with online batch learning

Even though S3WA is designed for chunk-based learning, we analyze its performance in online batch learning against self-evolving neural networks, i.e. ADL,Parsnet, and DEVDAN. We adapted the proposed algorithm to such a scenario, so that we have direct and meaningful comparisons between these models. We standardized the batch size to 200 across all models to ensure that any performance differences were due to the models inherent characteristics.

Fig. 7a presents the critical difference diagram of Nemenyi post-hoc statistical test on prequential accuracy across all data streams. The Nemenyi critical difference diagram is a visual representation used to compare the performance of different statistical methods or machine learning algorithms based on a specific metric, such as prequential accuracy or computational time. This diagram is commonly applied in post-hoc statistical tests when multiple comparisons need to be made between groups of models, helping to identify which methods exhibit statistically significant differences from one another.

The diagram is constructed based on the mean rankings of the evaluated methods, which are derived from experiments conducted on different datasets. On the horizontal axis, each method is assigned a position according to its average ranking, where lower values indicate better relative performance. Above the axis, bars or lines connect methods that do not show statistically significant differences, based on the critical difference threshold determined by the Nemenyi test. If two methods are connected by this line, it means that the difference in their mean rankings is not large enough to be considered statistically relevant, implying that their performance is

Table 3 – Table of hyperparameter ranges for random search.

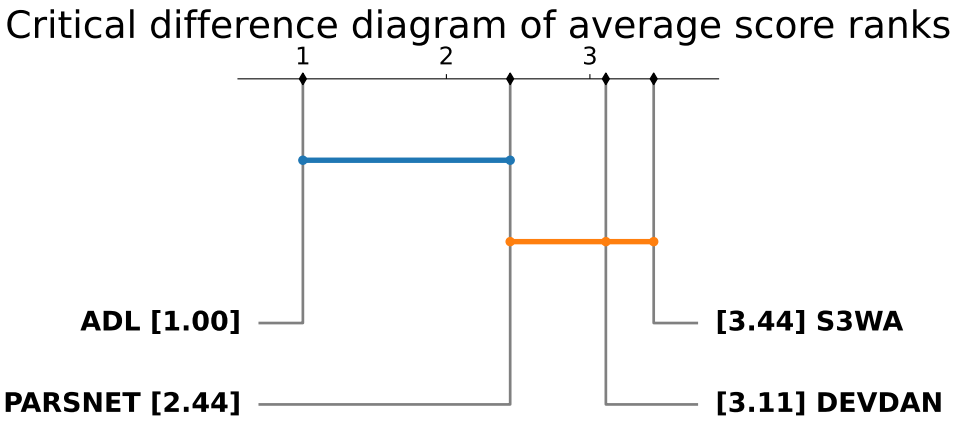
Hyperparameter	Probability distributions for randomized search
Ozabag-Adwin	
ensemble size	randint [1, 20]
DWM	
ensemble size	randint [1, 20]
β	uniform in [1^{-5}, 0.999]
θ	uniform in [1^{-5}, 0.999]
period	randint [1, 1000]
SRPC	
ensemble size	randint [1, 20]
subspace size	randint [1 , 100]
training method	choice {random subspaces, resampling, random patches}
subspace mode	choice {sqrtM1, MsqrtM1, percentage}
λ	uniform in [$1e^{-5}$, 10]
CSB2	
ensemble size	randint [1 , 20]
positive cost	uniform in [0, 1]
negative cost	uniform in [0, 1]
S3WA	
layers	randint [1, 500]
β_1	uniform in [0, 1]
β_2	uniform in [0, 1]
learning rate	uniform in [$1e^{-5}$, 1]
weight decay	uniform in [0, 1]
start	randint [2, 0.5*number of instances]

similar. Conversely, methods that are not connected by the same line have statistically distinct performances, indicating that one is superior to the other with a given confidence level.

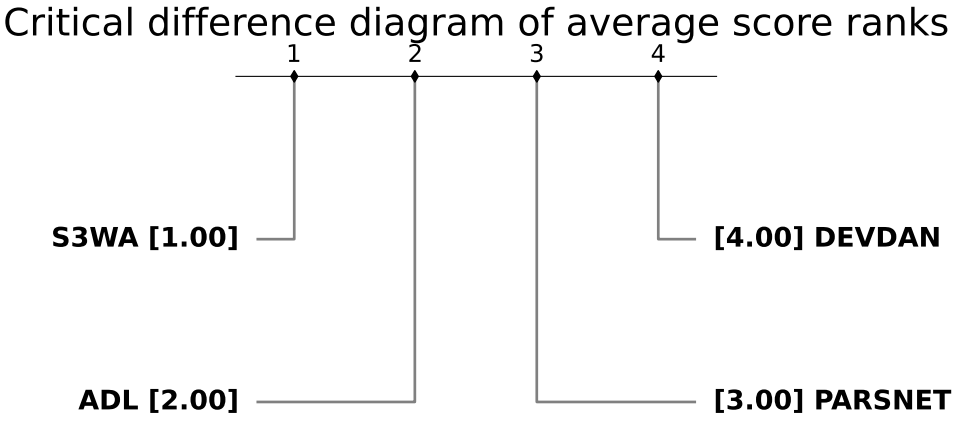
ADL and Parsnet have similar predictive performances. Parsnet, S3WA and DEVDAN are within the critical difference, i.e. the generalization of Parsnet, Devdan and S3WA are similar. This might indicate that adaptive architectures are beneficial when learning from batched data. This scenario requires larger weight updates for each batch. Such updates could be achieved with evolving architectures.

In Fig 7b, we show the critical difference diagram on computational time of the compared methods. All algorithms are statistically different from each other. S3WA is the most efficient, followed by ADL, PARSNET and DEVDAN. Self-evolving architectures require additional mechanisms to detect concept drifts, create and remove nodes and layers. These mechanisms have a computational overhead that can harm potential solutions for high-speed data. Unlike the others models, S3WA maintains a constant semi-supervised network architecture, which is more efficient and suitable for time sensitive applications. It is important to highlight that, despite having a constant architecture, S3WA could deliver similar performance to two self-evolving algorithms (ParsNet and DEVDAN) with significantly less computational time in a learning

Figure 7 – Critical difference diagram of Nemenyi post-hoc statistical test across all data streams. Values in brackets are mean ranks.



(a) Prequential accuracy.



(b) Computational time.

Source: Author

scenario to which it was not designed. This result was achieved by the use of an EMA for weight updates, producing competitive generalization with statistically less time overhead.

The results presented in Table 4 show the processing times for each model using 5% labeled data in a batch processing scenario. The S3WA model demonstrated superior computational efficiency, particularly for the STAGGER1 and STAGGER2 data streams, with processing times of 0.21 and 0.22 seconds, respectively. This efficiency can be attributed to the model's

Table 4 – Processing time (in seconds) for each method on each data stream.

		DEVDAN	ADL	Parsnet	S3WA
Abrupt	STAGGER1	35,74	0,48	3,27	**0,21**
	STAGGER2	35,77	0,49	3,82	**0,22**
	SEA1	16,21	0,63	2,98	**0,28**
	SEA2	22,54	0,63	2,91	**0,29**
	Agrawal1	168,84	0,62	4,11	**0,28**
	Agrawal2	241,91	0,76	4,98	**0,33**
	Agrawal3	172,23	0,62	4,05	**0,27**
	Agrawal4	87,32	0,62	3,8	**0,27**
	Sine1	12,14	0,37	1,8	**0,17**
	Sine2	25,89	0,61	2,97	**0,27**
Gradual	STAGGER1	35,58	0,49	2,93	**0,2**
	STAGGER2	35,77	0,49	3,82	**0,21**
	SEA1	19,84	0,63	4,31	**0,27**
	SEA2	21,03	0,62	3,56	**0,28**
	Agrawal1	172,36	0,62	4,17	**0,28**
	Agrawal2	246,41	0,76	5,14	**0,33**
	Agrawal3	172,71	0,62	4,50	**0,27**
	Agrawal4	131,6	0,62	4,53	**0,28**
	Sine1	12,12	0,37	1,91	**0,16**
	Sine2	25,38	0,6	3,06	**0,28**
Real	NOAA	N/A	0,5	53,39	**0,23**
	Sensor	N/A	3,37	2573,4	**1.67**
	Power	60,46	0,76	85,32	**0,42**
	Electricity	74	1,58	172,95	**0,45**
	Chess	0,27	0,04	0,15	**0,008**
	Luxembourg	N/A	0,08	0,25	**0,021**

Source: Author

streamlined and optimized architecture, which allows for faster and more effective processing of data in batches.

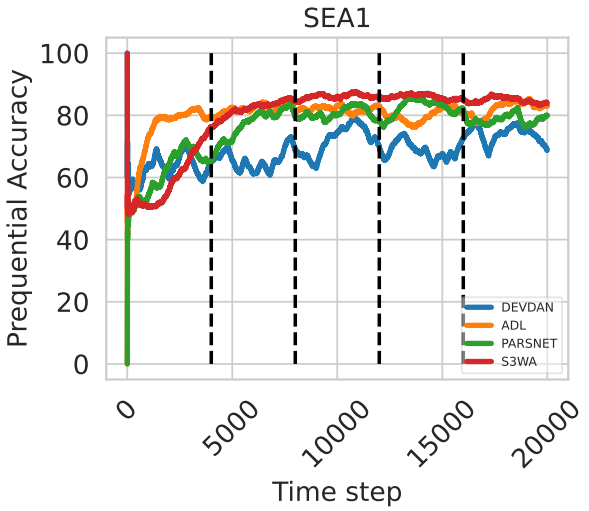
In comparison, the DEVDAN model exhibited significantly longer processing times, especially for the Agrawal data streams. While DEVDAN benefits from an evolving structure that enhances its adaptive capabilities, this feature also introduces higher computational costs, making it less efficient for data streams where timely performance is critical.

Another notable aspect is the consistent performance of S3WA across different data streams, regardless of their complexity. While models like ParsNet and ADL displayed greater variability in processing times, S3WA consistently achieved low and predictable processing times. This reliability positions it as a robust solution for scenarios requiring batch processing, ensuring efficient use of computational resources.

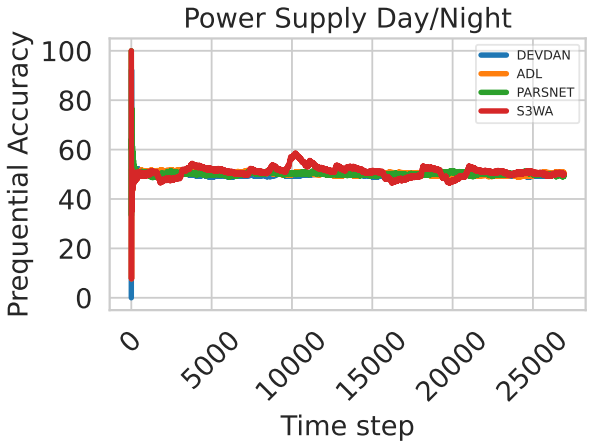
Fig. 8 shows the prequential accuracy of the best replicate for each method on the SEA1 and Power Supply Day/Night streams. S3WA outperforms the other techniques before and

Figure 8 – Plots of prequential accuracy produced by the best run of each of the methods for artificial data streams with abrupt concept drifts. Dashed lines denote concept drifts.

(a)



(b)



Source: Author

after drifts on the SEA1 stream. In the Power Supply Day/Night stream, S3WA outperforms DEVDAN, ADL, and PARSNET. It offers competitive performance and superior time efficiency, which is ideal for real-time applications requiring fast model updates. With the proper selection of the α hyperparameter, S3WA can react to concept drifts by balancing the influence of new weights and older solutions via an EMA. This implicit adjustment mechanism handles both abrupt and gradual changes, ensuring robust performance in dynamic environments.

Online batch learning is particularly challenging for S3WA as these batches may produce large weight updates, which can cause a noisy EMA. This can compromise the effectiveness of the weight averaging mechanism, making it difficult to find large optima and, therefore, maintain consistency throughout updates. Each μ update may become unrelated to previous time steps. The model should retain knowledge learned in the past to produce a good solution μ, but this relationship is lost when each batch is formed of completely new examples, leading to

performance degradation. These challenges are evident in our experiments, where S3WA did not always achieve superior accuracy compared to ADL, Parsnet, and DEVDAN. The EMA's inconsistency with discontinuous data blocks can cause S3WA to miss large optima, impacting its ability to adapt and maintain high accuracy. ADL, Parsnet, and DEVDAN are specifically designed for adapting their architecture to represent relatively large portions of data, which may explain their superior performance in this context.

Although S3WA may not always outperform these techniques in predictive performance, it offers significantly better computational efficiency. This makes S3WA ideal for applications where computational time is critical (e.g. high-speed data streams) and relatively high accuracy is sufficient. Its architectural simplicity and low computational cost are major advantages in online learning environments requiring adaptability and real-time efficiency. Unlike S3WA, which processes data in real-time, ADL, Parsnet, and DEVDAN use batch processing, which can hinder the model's ability to adapt to rapid data changes and introduce delays in response time since updates are not made instantaneously.

Self-evolving methods have high computational costs due to dynamic adaptations, whilst S3WA is more efficient. Although batch processing can disrupt the EMA and affect S3WA's performance, it still offers a practical and effective alternative to self-evolving architectures, providing robust performance with reduced computational demands.

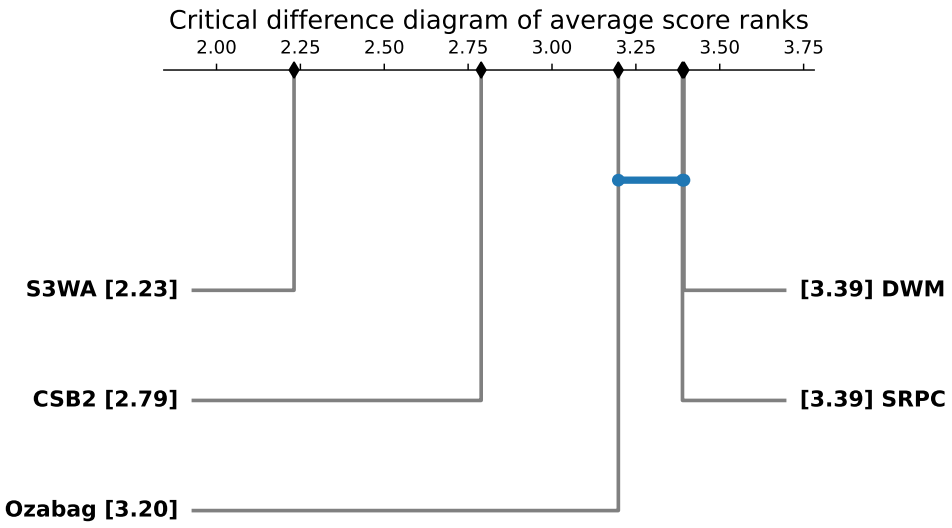
5.1.3 Experiments with chunk based learning

For all models, we use $N = 200$. In supervised ensembles, the entire data stream was used for the test, while only the labeled data was used for the training, in addition, a neural network was used as a base learner for the ensembles.

S3WA was significantly superior to all other techniques, as evidenced by the statistical tests performed on both prequential accuracy (p-value = 4.11×10^{-37}) and computational time (p-value = 7.08×10^{-164}). These results highlight the model's consistent ability to outperform its competitors across multiple metrics. Figure 9 provides a critical difference diagram that visually represents the statistical comparisons, illustrating the clear advantage of S3WA. With the best average rank of 2.23, S3WA surpassed other ensemble methods, including CSB2, Ozabag, DWM, and SRPC.

The exceptional performance of S3WA can be attributed to its ability to efficiently integrate unlabeled data into its learning process, a feature that sets it apart from traditional supervised ensemble methods. This integration enhances the model's ability to generalize, ensuring robust performance across diverse datasets while maintaining high predictive accuracy. Unlike CSB2 and DWM, which often rely heavily on labeled data, S3WA leverages its semi-supervised learning mechanism to extract meaningful patterns even from unlabeled examples. This not only reduces the dependency on labeled datasets but also makes S3WA particularly

Figure 9 – Critical difference diagram of Nemenyi post-hoc statistical test on mean prequential accuracy across all data streams. Values in brackets are mean ranks.



Source: Author

suitable for real-world scenarios where labeled data is sparse or costly to obtain.

In Tab. 5, we present the mean prequential accuracy across all data streams, highlighting the effectiveness of the S3WA model in various scenarios. S3WA demonstrated exceptional performance in several data streams with abrupt concept drift, such as STAGGER1 and STAGGER2, achieving accuracy rates exceeding 98%. This result underscores the model's robustness in adapting to sudden changes in data distributions, leveraging its semi-supervised learning mechanism and stochastic weight averaging to maintain high predictive performance even in dynamic environments.

For real-world data streams, S3WA also exhibited remarkable accuracy. In the Sensor data stream, it significantly outperformed DWM, CSB2, OZABAG, and SRPC. This achievement reflects S3WA's ability to integrate labeled and unlabeled data effectively, enhancing generalization and enabling it to adapt to the nuanced patterns often present in real-world datasets. The model's capacity to handle both abrupt and gradual concept drift likely contributed to its superior performance in this context.

In the NOAA and Electricity data streams, which are well-known benchmarks for evaluating online learning models, S3WA outperformed all other methods. These datasets typically feature non-stationary behaviors and recurring patterns, posing challenges for many models. S3WA's chunk-based processing and adaptive weight averaging allowed it to maintain consistency and accuracy over time, showcasing its suitability for these complex, real-world tasks.

Table 5 – Mean of prequential accuracy for all methods.

Drift types	Data streams	OZABAG	DWM	CSB2	SRPC	S3WA
Abrupt	STAGGER1	82,30 ± 0,91	86,87 ± 0,88	86,68 ± 0,98	50,41 ± 4,44	**98,83 ± 0,06**
	STAGGER2	89,50 ± 0,54	88,95 ± 0,70	89,88 ± 1,18	50,66 ± 8,08	**98,73 ± 0,06**
	SEA1	50,59 ± 1,68	50,61 ± 1,56	50,95 ± 1,69	47,67 ± 9,36	**69,79 ± 2,66**
	SEA2	**83,38 ± 0,40**	81,91 ± 0,95	82,16 ± 0,44	48,54 ± 10,39	71,01 ± 2,67
	Agrawal1	48,80 ± 0,01	48,75 ± 0,01	**49,96 ± 0,18**	49,89 ± 0,70	46,12 ± 24,70
	Agrawal2	49,10 ± 0,03	48,98 ± 0,04	49,31 ± 0,05	50,25 ± 0,80	**54,23 ± 28,59**
	Agrawal3	48,95 ± 0,04	48,77 ± 0,03	49,08 ± 0,03	**49,81 ± 0,73**	44,48 ± 22,68
	Agrawal4	48,84 ± 0,03	48,73 ± 0,02	49,05 ± 0,03	49,21 ± 0,82	**49,35 ± 21,03**
	Sine1	48,95 ± 0,23	48,46 ± 0,23	49,22 ± 0,15	49,73 ± 3,18	**69,55 ± 2,75**
	Sine2	49,45 ± 0,09	49,41 ± 0,11	49,98 ± 0,01	N/A	**69,12 ± 3,33**
Gradual	STAGGER1	86,03 ± 0,61	85,00 ± 0,80	88,13 ± 0,55	50,44 ± 4,09	**97,95 ± 0,06**
	STAGGER2	88,22 ± 0,49	90,31 ± 0,53	88,83 ± 0,89	51,22 ± 6,15	**97,67 ± 0,05**
	SEA1	50,90 ± 0,91	49,55 ± 1,89	51,62 ± 1,69	52,18 ± 9,69	**71,15 ± 2,29**
	SEA2	**82,03 ± 0,40**	80,63 ± 0,97	81,10 ± 0,54	48,50 ± 8,33	70,66 ± 2,03
	Agrawal1	48,80 ± 0,02	48,73 ± 0,01	49,92 ± 0,23	**50,43 ± 0,77**	48,48 ± 3,44
	Agrawal2	49,12 ± 0,05	48,93 ± 0,05	49,30 ± 0,04	**50,22 ± 0,90**	46,31 ± 4,52
	Agrawal3	49,06 ± 0,03	48,84 ± 0,03	49,18 ± 0,04	**50,25 ± 0,58**	48,57 ± 3,74
	Agrawal4	48,84 ± 0,04	48,69 ± 0,02	49,01 ± 0,04	**50,00 ± 0,79**	48,25 ± 3,69
	Sine1	48,96 ± 0,27	48,56 ± 0,21	48,92 ± 0,16	50,15 ± 2,04	**69,15 ± 2,39**
	Sine2	47,60 ± 0,08	49,71 ± 0,14	49,97 ± 0,03	N/A	**68,35 ± 3,31**
Real	NOAA	60,64 ± 3,37	60,44 ± 0,12	61,61 ± 0,53	54,90 ± 17,60	**64,60 ± 0,36**
	Sensor	54,34 ± 0,20	54,87 ± 0,55	53,97 ± 0,32	50,09 ± 1,43	**72,59 ± 0,93**
	Power	49,74 ± 0,20	50,03 ± 0,08	49,40 ± 0,20	50,02 ± 1,81	**50,50 ± 0,25**
	Electricity	57,78 ± 0,35	58,71 ± 0,21	57,91 ± 0,41	47,76 ± 8,73	**59,79 ± 0,02**
	Chess	52,66 ± 3,37	**53,33 ± 3,26**	52,75 ± 3,87	50,06 ± 10,35	50,66 ± 2,02
	Luxembourg	**57,11 ± 1,67**	55,51 ± 0,51	56,64 ± 1,05	47,76 ± 3,83	49,04 ± 1,68

Source: Author

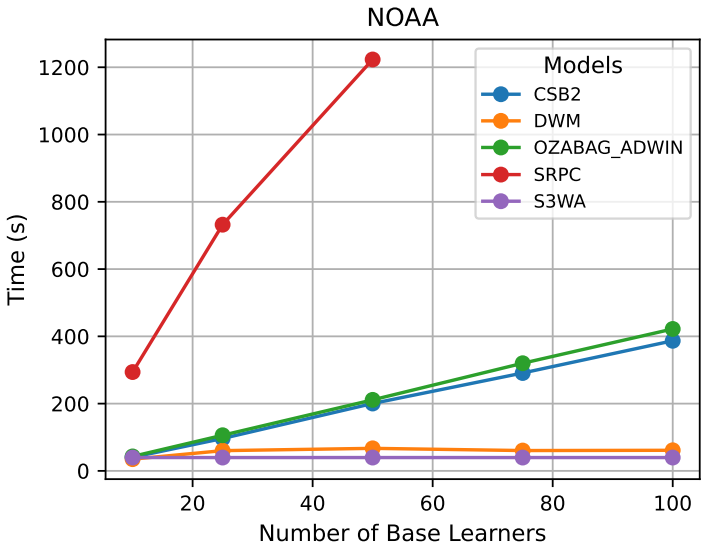
When analyzing the Power Supply data stream, S3WA achieved comparable accuracy to DWM and OZABAG, outperforming SRPC and CSB2. This result suggests that while S3WA excels in most scenarios, certain competitive ensemble models, such as DWM and OZABAG, can achieve similar performance in specific contexts. Nonetheless, S3WA's balance between computational efficiency and predictive accuracy gives it an edge, particularly when considering its overall performance across all data streams.

In Chess and Luxembourg data streams, S3WA achieved accuracy comparable to SRPC but slightly lower than DWM, CSB2, and OZABAG. This performance highlights potential areas for improvement, particularly in data streams with specific characteristics or distribution patterns that may favor alternative ensemble techniques. Despite this, S3WA's ability to remain competitive across a diverse range of data streams emphasizes its versatility and general effectiveness.

Fig. 10 presents the computational time of each method with respect to the number of base learners on the NOAA data stream. As the number of base learners increases, the SRPC method exhibits a steep rise in computational time, becoming significantly slower. In contrast, the CSB2, DWM, and Ozabag-Adwin models demonstrate a more gradual increase, indicating better scalability. However, it is noteworthy that SRPC could not handle more than 75 base learners, highlighting a clear limitation in the scalability of this approach, especially in scenarios requiring high computational capacity.

On the other hand, S3WA stands out for its independence from base learners, functioning

Figure 10 – Computational time of numbers of base learners



Source: Author

as a single model. This unique characteristic eliminates the overhead associated with managing ensembles, enabling S3WA to maintain superior performance in chunk-based learning scenarios. Despite its significantly lower computational time, S3WA consistently delivers competitive and robust results, outperforming ensemble-based methods in multiple aspects.

The lack of dependency on base learners provides S3WA with a notable advantage in terms of scalability and computational efficiency. While methods like SRPC face substantial challenges as the number of base learners increases, S3WA sustains its efficiency even in complex and non-stationary data streams. This makes it a more robust and suitable solution for real-time applications, where computational resources and processing time are critical constraints.

5.2 Results with JIT-SDP datasets

5.2.1 Experimental Method

In 2008, Kim et al. laid the groundwork for JIT-SDP by identifying and describing a set of input features to be used in such models (KIM; WHITEHEAD; ZHANG, 2008). Later, Kamei et al. performed a large-scale empirical analysis, examining 14 features grouped into five categories: diffusion, size, purpose, history, and experience. Their study demonstrated that these factors are effective predictors of defect-inducing software changes, achieving high performance across both open-source and commercial projects (KAMEI et al., 2012). Shihab et al. complemented these findings by showing that certain metrics such as the number of added lines of code, the proportion of bug fixes relative to all file changes, the quantity of bug reports linked to a commit, and developer experience are strong indicators of defect-prone changes (SHIHAB et al., 2012).

Table 6 – The number of bug-inducing and clean commits in the projects of the ApacheJIT

Project	**Bug-Inducing**	**Clean**	**Total**
ActiveMQ	1.404 (23%)	4.722	6.126
Camel	3.708 (14%)	19.622	22.700
Cassandra	3.117 (38%)	5.042	8.159
Flink	2.811 (24%)	8.880	11.691
Groovy	1.614 (20%)	6.445	8.059
Hbase	3.782 (43%)	4.948	8.730
Hive	4.223 (63%)	2.619	6.842
Ignite	2.439 (20%)	9.597	12.036
Kafka	838 (15%)	4.995	5.833
Spark	632 (43%)	833	1.465
Zeppelin	622 (42%)	829	1.451
Zookeeper	342 (40%)	497	839
Total	**25.532 (27%)**	**68.399**	**93.931**

In this study, we used ApacheJIT, one of the largest and most robust datasets available for JIT-SDP. This dataset was designed to meet the requirements of machine learning and deep learning, comprising a total of 93.931 commits from 12 widely used Apache projects, named as Activemq, Camel, Cassandra, Flink, Groovy, Hbase, Hive, Ignite, Kafka, Spark, Zeppelin and Zookeeper. Of these commits, 25.532 were labeled as bug-inducing, and 68.399 as clean. The data spans over a decade (2003 to 2019), providing a rich history of code changes. In the Table 6 shows the statistics of dataset ApacheJIT. The dataset are publicly available [1].

The construction of ApacheJIT follows rigorous filtering criteria, applying a modified SZZ algorithm to identify defective commits. The methodology includes steps such as collecting bug reports, analyzing commits related to these bugs, and applying a series of heuristics to eliminate false positives and negatives. Key exclusion criteria include removing large-scale commits (modifications in more than 100 files or 10,000 lines), trivial commits (modifications involving only comments or whitespace), and those outside the primary language scope (Java). These steps ensure the quality and relevance of the data for defect prediction studies.

Although ApacheJIT, in its entirety, provides a comprehensive overview of various code changes, this work opted to use data from individual projects rather than the complete dataset. This approach allows us to explore the specificities of each project and evaluate how models perform in distinct contexts. Each project presents unique characteristics that influence both defect patterns and extracted metrics.

In addition to defect labels, ApacheJIT includes detailed metrics on the code changes, which are widely used in JIT-SDP models. We consider 12 metrics grouped in four dimesions derived from the repository data of a project (KAMEI et al., 2012). Each metric is treated as a feature.

[1] https://doi.org/10.5281/zenodo.5907001

Size dimesion: This dimension focuses on the magnitude of changes made to the code. Its consists of two metrics, `Number of lines added and Number of lines deleted`, indicate, respectively, the number of lines of code added or deleted in the commit. These metrics have been identified as relevant, as larger changes are more likely to introduce defects.

Diffusion dimesion: Here, the scope of a change across the project is assessed. `Number of files touched` measures the total number of files modified in the commit. `Number of directories touched` reflects the total number of directories impacted by the changes. `Number of subsystems touched` indicates the total number of subsystems (modules or components) affected by the commit. `Change entropy` is a metric that reflects the dispersion of changes across modified files, calculated as a measure of uncertainty. With this, diffusion metrics are strong defect predictors because highly dispersed changes are more challenging to test and review.

History dimesion: This dimension delves into the past modifications and activity of files. `Number of distinct developers touched files` represents the number of developers who made changes to the files in the commit, measuring collaboration. `Average time from last change` captures the average time, in days, since the last modification of the files involved. `Number of unique changes in files` counts the unique changes made to the files in the commit. `Change date` indicates when the commit was made, enabling temporal analysis of the change history. This metrcis showing that files frequently modified are more likely to contain defects.

Experience dimesion: This dimension evaluates the expertise of the developer responsible for the changes. `Change author experience` quantifies the author's experience based on their history of changes in the project. `Change author recent experience` focuses on changes made by the author within a recent timeframe. `Change author subsystem experience` captures the author's accumulated experience in the subsystem where the change occurred. Developer experience significantly impacts code quality, experienced developers tend to commit fewer errors.

These metrics enable a deep analysis of the characteristics of code changes and provide valuable inputs for supervised and unsupervised learning models.

To ensure methodological consistency, the hyperparameter tuning was conducted exactly as described in Subsection 5.1.1. Similarly, the prequential accuracy was computed using the same procedure and forgetting factor.

5.2.2 Experiments with online batch learning

Figure 11 presents graphs illustrating the evolution of the prequential accuracy of the evaluated methods over time in two distinct scenarios represented by the Spark and Zeppelin

datasets. To reduce variability and make the accuracy trend more visible over time, we applied an Exponential Moving Average (EMA) with a smoothing factor of $\alpha = 0.005$. The EMA is computed recursively, where each smoothed point is a weighted combination of the current value and the previous smoothed value. This method is widely used for smoothing time series, as it maintains a quick response to data changes while reducing short-term fluctuations.

These graphs compare the performance of different models in terms of adaptability in a continuous learning environment, where new data arrives incrementally and may exhibit statistical variations known as concept drift.

In Figure 11a, significant variability in the Spark can be observed, requiring the models to exhibit greater adaptability to changes in data patterns. In this scenario, the ADL and Parsnet methods demonstrated resilience, maintaining or even improving their performances during periods of instability. Conversely, S3WA and DEVDAN exhibited more pronounced drops in accuracy, especially during moments of higher instability, highlighting their limitations in handling concept drift.

In Zeppelin dataset, Figure 11b reveals considerable variations in accuracy over time, challenging the assumption that this scenario would be more stable. The observed fluctuations indicate the presence of concept drift, albeit in a manner distinct from that in the Spark dataset. In this environment, although ADL and Parsnet continued to demonstrate superior performance, S3WA struggled significantly, highlighting that its chunk-processing-based architecture is less effective when adapted to batch processing scenarios, as used in this experiment.

These results underscore the importance of aligning model design with the characteristics of the data and the processing method employed to maximize the efficiency and effectiveness of predictions in continuous learning scenarios.

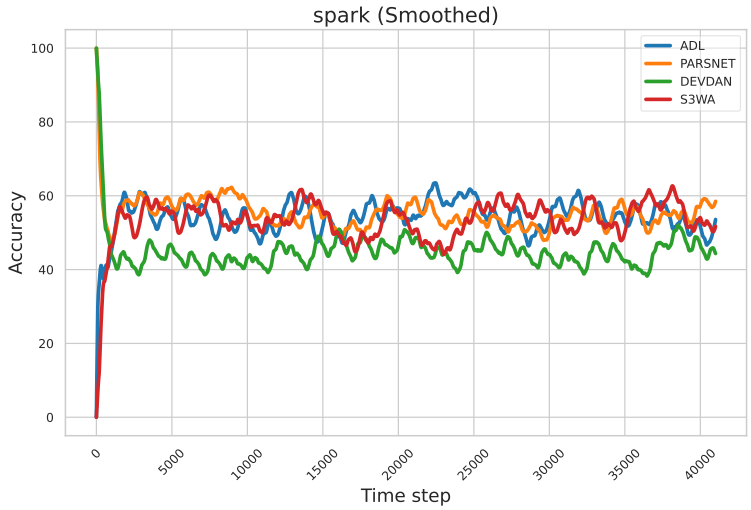
The Table 7 compares the prequential accuracy of the online ML models. The DEVDAN model displays inconsistent performance across different software projects, achieving notably high results in projects like Camel and Ignite. This suggests a strong ability of this model to adapt to the specificities and variations in these contexts, thanks to its capability to learn incrementally and adjust its structure dynamically. However, its limitations are evident in projects like Kafka and Spark, where it struggles with more complex defect patterns or unpredictable data.

On the other hand, the ADL model exhibits high accuracy in most projects, except for Spark, where its performance is only mediocre. This model excels due to its ability to adapt its layers and neurons throughout the learning process, which appears particularly effective in capturing complex patterns in varied environments, justifying its predominance in several analyzed contexts.

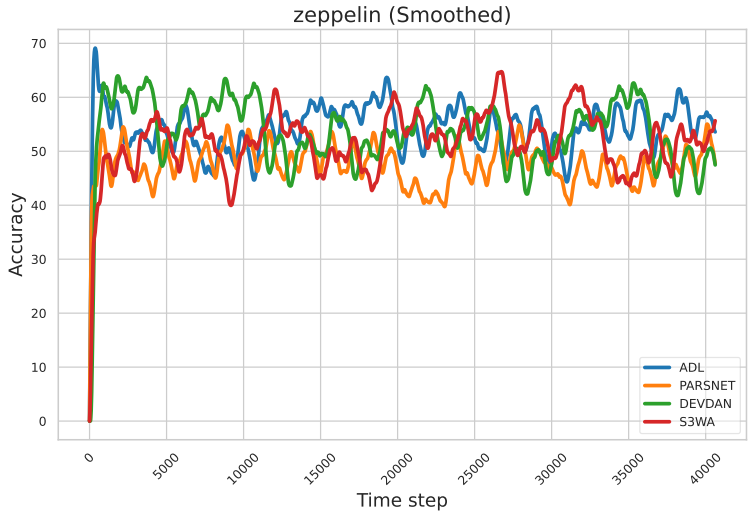
Parsnet, focusing on parsimonious neural networks, shows excellent results in projects like Camel and Ignite, outperforming the other models. Its efficiency can be attributed to a structure that allows for deep and efficient learning with fewer parameters, facilitating quick

Figure 11 – Plots of prequential accuracy produced by the best run of each of the methods.

(a)



(b)



Source: Author

adaptation to changes in data patterns. However, its performance is less impressive in Kafka, indicating potential challenges in dealing with unusually complex defect patterns or highly noisy data.

Finally, the S3WA, which shows the lowest prequential accuracy in projects like Cassandra and Flink. While S3WA performs well in scenarios with consistent data streams, its architecture struggled to adapt to the complexities of the batch processing framework employed in this study.

Table 8 reveals the processing times in seconds for the DEVDAN, ADL, Parsnet, and S3WA models, applied to different datasets. This analysis of processing times provides insights into the operational efficiency of each model in specific batch processing contexts, even though some are optimized for other modes of operation.

Table 7 – Mean of prequential accuracy for all methods.

	DEVDAN	ADL	Parsnet	S3WA
Activemq	N/A	72,87	**73,90**	60,41
Camel	83,4	84,37	**85,69**	80,98
Cassandra	62,4	**63,14**	62,09	52,65
Flink	75,64	**78,25**	77,08	68,05
Groovy	N/A	77,05	**78,29**	54,52
Hbase	N/A	**62,01**	60,38	50,00
Hive	55,37	58,15	**65,23**	53,10
Ignite	N/A	77,77	**81,96**	76,27
Kafka	N/A	**55,19**	47,94	48,28
Spark	45,20	50,78	**57,33**	50,91
Zeppelin	56,11	**65,21**	61,65	54,47
Zookeeper	49,39	**71,61**	53,80	51,89

Source: Author

Table 8 – Processing time (in seconds) for each method on each data stream.

	DEVDAN	ADL	Parsnet	S3WA
Activemq	N/A	0,247	8,277	0,455
Camel	9,31	0,660	115,0	1,673
Cassandra	3,293	0,223	15,34	0,583
Flink	4,753	0,373	32,07	0,851
Groovy	N/A	0,292	15,07	0,583
Hbase	N/A	0,280	15,98	0,693
Hive	2,887	0,230	11,06	0,489
Ignite	N/A	0,308	33,89	0,887
Kafka	N/A	0,081	1,290	0,153
Spark	0,537	0,062	0,763	0,089
Zeppelin	0,603	0,057	0,836	0,095
Zookeeper	0,312	0,035	0,442	0,046

Source: Author

DEVDAN, applied in scenarios like Camel, shows variable times, being relatively slower compared to other models such as ADL. This longer processing time can be attributed to the dynamic nature of DEVDAN, which continuously adjusts and recalculates its parameters in response to new data. While this feature is beneficial for adapting to changes in data, it increases processing complexity.

In contrast, ADL demonstrates remarkable efficiency, with the shortest processing times in almost all projects. ADL's optimized architecture allows for quick and effective adaptations of its layers without the need for extensive recalibrations, facilitating more agile processing that is ideal for the batch format used in tests.

Parsnet, despite its ability to learn efficiently with fewer parameters, records the longest

processing times in many of the projects. This reflects the internal complexity of the model, which performs intensive parsimonious adjustments during training. Such an approach results in slower processing, especially in batch scenarios where continuous optimization can be more time-consuming.

Finally, S3WA, despite being designed for chunk processing with sliding windows, adapts to the batch format with competitive processing times, though not as fast as ADL. Adapting its mechanism to a batch scenario requires extra effort to manage the complexity of the sliding windows and integrate data non-continuously. Although efficient, the nature of this adjustment results in a slight disadvantage in speed compared to models more aligned with batch processing, like ADL.

5.2.3 Experiments with chunk based learning

This subsection presents the experiments conducted using the chunk processing learning method, focusing on the evaluation of the S3WA model in its optimal processing format. Unlike batch learning, chunk processing allows data to be processed continuously in smaller, adjustable blocks, better reflecting the dynamic nature of data streams in continuous learning scenarios, such as JIT-SDP.

Table 9 illustrates the prequential average accuracy of the S3WA model when applied using its preferred processing method, in chunks, and offers an interesting perspective compared to Table 7, which presents the results of S3WA in a batch processing environment along with other models such as DEVDAN, ADL, and Parsnet. It is observed that the S3WA, when operating in chunks, exhibits a substantial improvement in its results compared to batch processing. For example, while in batch the S3WA had lower results on projects Camel and Ignite, in chunk processing, it achieves significantly higher accuracies, 87.23 and 81.89, respectively.

This difference in results is even more notable when considering the performances of other models in Table 7. Models like ADL and Parsnet, which excelled in batch, tend to surpass S3WA in such a scenario, but it is evident that the S3WA regains its competitiveness when applied in its ideal scenario. This phenomenon can be attributed to the efficiency of the S3WA in dealing with continuous data streams, where its adaptation capabilities are better explored and where the integration of new data occurs more fluidly and naturally.

Furthermore, even in challenging projects like Kafka, where all models tend to perform lower, the S3WA in chunks shows a comparative improvement, indicating that its architecture is particularly advantageous for dealing with the variations and complexity of real-time data. These results suggest that the choice of the model's processing method is not just a technical issue, but a strategic decision that can directly influence the effectiveness of the model in practical applications. Thus, while models like ADL and Parsnet may be preferable in batch scenarios for certain types of data, the S3WA emerges as a robust option when data can be processed in its

Table 9 – Mean of prequential accuracy for S3WA.

Project	Batch	Chunk
Activemq	60,41	**76,70**
Camel	80,98	**87,23**
Cassandra	52,65	**72,71**
Flink	68,05	**79,92**
Groovy	54,52	**60,00**
Hbase	50,00	**72,60**
Hive	53,10	**72,45**
Ignite	76,27	**81,89**
Kafka	48,28	**55,35**
Spark	50,91	**65,38**
Zeppelin	54,47	**61,48**
Zookeeper	51,89	**55,66**

Source: Author

most natural and continuous form, thereby maximizing the model's capabilities.

5.3 Discussion

The experiments conducted in this research provides a comprehensive view of model performance in non-stationary learning environments, covering both artificial data streams and real-world applications of JIT-SDP. The analysis of the results allows us to identify behavioral patterns in the tested models and understand the main challenges faced in scenarios where data is dynamic and subject to continuous change.

Experiments in non-stationary environments demonstrated that conventional online learning models exhibit varying degrees of sensitivity to concept drift, depending on the strategy used for model updates. Models such as ADL, ParsNet, and DEVDAN showed that online learning approaches can be effective in scenarios where changes occur gradually but struggle with abrupt drifts. For instance, ADL managed to respond to minor variations over time but suffered performance degradation when data patterns changed significantly. ParsNet, which employs a parsimonious learning approach, showed greater stability in smooth drift scenarios but had difficulty quickly reacting to structural changes. Meanwhile, DEVDAN, which features a dynamic network structure adjustment mechanism, performed well in adapting to evolving data streams but incurred high computational costs.

On the other hand, S3WA demonstrated a more balanced performance over time due to its integration of semi-supervised learning, SWA and EMA, which collectively improved its adaptability to non-stationary data. Unlike traditional models that struggle to accommodate rapid shifts in data distribution, S3WA leveraged EMA to smooth parameter updates, preventing abrupt fluctuations that typically degrade model performance in concept drift scenarios. The use

of EMA ensured that historical weight information was retained efficiently, stabilizing updates and enhancing the model's robustness against noise and sudden changes.

In the specific context of JIT-SDP, the experiments highlighted the challenges posed by the dynamic nature of commits and the necessity for models to adapt to evolving software development patterns. The main issue identified was the performance degradation of conventional supervised learning methods when applied to an environment where the data distribution continuously evolves. Traditional batch based learning models struggle to capture these changes as they operate under the assumption that patterns observed in historical data will remain valid in the future. However, since commit streams undergo modifications due to the introduction of new development practices, changes in programming teams, and tool updates, this assumption rarely holds in real-world scenarios.

The differences in processing times among the models highlight how the specific architecture and optimization of each can directly influence their effectiveness in different testing environments, emphasizing the importance of selecting the most suitable model based on both specific data processing needs and compatibility with the anticipated operational context.

Although batch processing may initially present good results, it fails to meet the real-world demands of scenarios such as JIT-SDP. While this method proves efficient in controlled tests, it is fundamentally inadequate for dynamic and non-stationary environments, where conditions constantly change and decisions must be made in real-time. Batch processing introduces significant delays, as models are only updated after accumulating a predetermined amount of data. This means that, between updates, the model operates with outdated information, making it incapable of capturing and responding to critical changes in data patterns, such as concept drift.

Furthermore, this approach disregards the natural flow of data in real-world scenarios, where information arrives continuously and incrementally. Using batch processing in such contexts not only limits the model's efficiency but also directly compromises its effectiveness, leading to an increase in errors and a reduction in its ability to predict defects in a timely manner. In situations where each incorrect decision can result in high financial costs, delays in development, or even security failures, relying on batch processing risks delivering subpar outcomes.

The chunk based learning approach used in the experiments proved to be more efficient than batch learning in dynamic scenarios, as it allows the model to gradually adapt to data changes without needing to process large volumes of information at once. Unlike batch learning, which requires a complete dataset before training, chunk based learning enables continuous model updates as new chunks are processed, making it more responsive to variations in commit streams. This characteristic is particularly advantageous in JIT-SDP, where commit patterns evolve over time, and a model trained using batch learning may quickly become obsolete. Additionally, dividing data into chunks allows for a better balance between stability and adaptability, reducing the need for extensive retraining and making the approach more efficient for continuous learning

scenarios.

S3WA emerged as a promising alternative for this scenario, as its semi-supervised learning mechanism facilitated better utilization of available data, reducing the need for fully labeled commits. This is particularly beneficial in JIT-SDP, where obtaining reliable labels can be a time consuming and uncertain process. Furthermore, the S3WA strategy helped smooth the model's parameter updates, ensuring greater stability in results over time. This feature was essential in handling the evolving commit patterns, allowing the model to maintain more consistent performance even in the presence of concept drift.

Another critical aspect analyzed was the impact of model processing time in the JIT-SDP context. Methods such as DEVDAN and ParsNet, although effective in adapting to data changes, have high computational costs, limiting their applicability in real-time scenarios. This occurs because these methods require frequent structural adjustments to the network, making the inference process slower and less efficient. S3WA, on the other hand, achieved a balance between computational cost and performance, making it a more viable alternative for applications in dynamic environments. This result reinforces the importance of developing models that are not only accurate but also computationally efficient, particularly in domains like JIT-SDP, where processing speed directly influences the model's practical utility.

From a statistical perspective, Nemenyi post-hoc tests showed that S3WA achieved competitive performance compared to other methods, excelling in moderate and recurrent concept drift scenarios. The mean ranking analysis revealed that the model maintained a more stable accuracy level, while other methods suffered significant degradation in specific data streams. This result reinforces the hypothesis that combining semi-supervised learning with adaptive regularization mechanisms can be an effective strategy for handling non-stationary environments.

In summary, the findings of this research demonstrate that combining semi-supervised learning techniques with concept drift adaptation strategies can be a viable solution to improve the effectiveness of models applied to dynamic environments. S3WA exhibited clear advantages over conventional methods in terms of both stability and computational cost, making it a promising approach for scenarios such as JIT-SDP.

6 Conclusion

In this master thesis, we addressed software defect prediction in the context of JIT-SDP, an essential area for ensuring software development quality. JIT-SDP enables a granular evaluation of code modifications, allowing for early defect detection and reducing the effort required for review and correction. However, challenges such as concept drift, class imbalance, and the need for adaptation to dynamic scenarios make its implementation complex.

The proposed model, S3WA, was developed to overcome these challenges by integrating semi-supervised learning with the Stochastic Weight Averaging technique. The experiments demonstrated that S3WA outperforms online models, by providing greater stability and accuracy in defect prediction, even in non-stationary scenarios. Additionally, its ability to leverage unlabeled data minimized the need for large volumes of annotated data, making the solution more practical and applicable in real-world environments.

The main innovation of S3WA lies in its ability to adapt to concept drifts through a weight-smoothing mechanism, ensuring more reliable predictions over time. The model also proved effective in identifying development patterns and accurately predicting defects across different commit profiles, reducing both false positives and false negatives.

Experiments with real-world datasets showed that S3WA not only delivered competitive accuracy compared to other state-of-the-art models but also demonstrated superior adaptability to dynamic data distributions. One of the key findings was the impact of processing strategies on model performance. While batch-based learning provided strong baseline results, it struggled to maintain stability in highly dynamic scenarios due to its dependence on periodic updates with fully labeled data. In contrast, the chunk-based approach, combined with a sliding window mechanism, allowed S3WA to adapt more effectively to non-stationary data streams. This adaptation was particularly beneficial in mitigating performance degradation caused by concept drift, ensuring that the model remained responsive to evolving software repositories.

Specifically, in the JIT-SDP context, S3WA proved to be an efficient and reliable alternative to conventional online learning models, particularly when using chunk-based learning. The results highlighted that chunk processing allowed S3WA to incrementally incorporate new commits while preserving relevant past information, balancing predictive performance and computational efficiency. Unlike batch processing, which required larger labeled datasets at fixed intervals, the chunk-based approach maintained real-time adaptability without incurring significant delays. Statistical analyses, including Nemenyi post-hoc tests, confirmed that S3WA, under chunk-based processing, consistently achieved superior prequential accuracy across multiple datasets. These findings reinforce the importance of selecting appropriate data processing strategies for software defect prediction and suggest that adaptive chunk-based approaches can

enhance the effectiveness of semi-supervised models in dynamic environments

In conclusion, this work advances the state of the art in JIT-SDP by proposing an adaptive, semi-supervised model for software defect prediction. S3WA represents a promising alternative for improving software quality in dynamic environments, reducing costs, and optimizing the development process.

6.1 Contributions

The main contribution of the master thesis lies in the incorporation of the weight averaging method to improve generalization in non-stationary data streams, without the high computational cost associated with self-evolving online learning, like ADL,DEVDAN, PARSNET, DWM and others. S3WA proves to be an efficient alternative to traditional supervised models, as its semi-supervised approach enables the integration of both labeled and unlabeled data in an effective manner, optimizing the use of available information and reducing the need for large volumes of annotated data. Additionally, the model offers advantages in terms of scalability and processing time, making it more suitable for real-time applications.

The experiments conducted indicate that the combination of weight averaging and unlabeled data significantly improves the model's generalization, allowing it to maintain high prequential accuracy over time, even in continuous data stream scenarios. The tests also demonstrate that the chunk-based learning approach used in S3WA contributes to more effective adaptation to concept drift, ensuring a more stable and robust response to variations in data patterns. Beyond its impact on software defect prediction, the results obtained reinforce the relevance of semi-supervised learning techniques and weight averaging to enhance the performance of continuous learning models across various domains. Thus, this study contributes to the advancement of the state of the art in machine learning applied to data streams, providing directions for future research in the field.

During the master degree, some papers and other collaborations were developed. This master's thesis was developed under the project "Aprendizado semissupervisionado online para predição de mudanças críticas em software", a partnership between UFPA, Federal Rural University of Pernambuco (UFRPE), University of Birmingham, and the company FlowUp, funded by the National Council for Scientific and Technological Development (CNPQ). The main target of the project was to develop a tool for supporting JIT-SDP.

The S3WA algorithm and the results shown at Section 5.1 were presented at The International Conference on Neural Information Processing (ICONIP) 2024. The corresponding paper "A weight averaging neural network for semi-supervised data stream learning" will be published in the Lecture Notes in Computer Science in 2025. As a non-first author, the Master candidate collaborated on another paper presented in ICONIP 2024 titled "Self-evolving optimization for data stream learning". The proceedings of the conference will be published in the

Communications in Computer and Information Science in 2025.

Also related to this thesis, the author published and presented the paper "Enhancing Software Quality: A Multicriteria Decision-Making Approach for Identifying Defective Modules" in the Latin America Conference on Computational Intelligence (LA-CCI) 2024. This paper was a first approach to the SDP problem, and it gave the author know-how for further development of the S3WA model.

It is worth highlighting the collaboration with the Federal Rural University of Pernambuco and the company FlowUp in the development of the Learning from commits tool, a result of the research project. This experience was important since it allowed the author to see how a JIT-SDP tool works in practice.

The author published other non-related papers during its master's degree. As first author, in 2023, the paper "Analysis of Twitter Posts on COVID-19 Vaccines in Brazil" was published in the Cadernos do IME, Série Informática. The journal is qualified with Qualis B3 by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES, Coordination for the Improvement of Higher Education Personnel). As co-author, in 2024, the paper "Web Service for Data Imputation in Univariate Time Series" was published in the 15th Workshop of Applied Computing for the Management of the Environment and Natural Resources (WCAMA).

6.2 Future Work

For future work, we propose implementing automated hyperparameter tuning mechanisms to enhance the adaptability of S3WA to different datasets and changing environments, reducing the need for manual intervention in model optimization. Additionally, we suggest adapting S3WA for embedded systems and IoT applications, where real-time defect prediction can significantly improve reliability and efficiency. This adaptation would require optimizing the model for low-power computing environments and ensuring it operates efficiently with constrained computational resources while maintaining high predictive performance.

Referências

AGRAHARI, S.; SINGH, A. K. Concept drift detection in data stream mining: A literature review. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, v. 34, n. 10, p. 9523–9540, 2022. Citado 2 vezes nas páginas 7 and 8.

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: A performance perspective. *IEEE transactions on knowledge and data engineering*, IEEE, v. 5, n. 6, p. 914–925, 1993. Citado na página 37.

ASHFAHANI, A.; PRATAMA, M. Autonomous deep learning: Continual learning approach for dynamic environments. In: SIAM. *Proceedings of the 2019 SIAM international conference on data mining*. [S.l.], 2019. p. 666–674. Citado na página 18.

ASHFAHANI, A. et al. Devdan: Deep evolving denoising autoencoder. *Neurocomputing*, Elsevier, v. 390, p. 297–314, 2020. Citado 2 vezes nas páginas 17 and 26.

BARTZ-BEIELSTEIN, T. Introduction: From batch to online machine learning. In: *Online Machine Learning: A Practical Guide with Examples in Python*. [S.l.]: Springer, 2024. p. 1–11. Citado na página 9.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of machine learning research*, v. 13, n. 2, 2012. Citado na página 39.

BIFET, A. et al. New ensemble methods for evolving data streams. In: *ACM SIGKDD*. [S.l.: s.n.], 2009. p. 139–148. Citado na página 14.

BISONG, E.; BISONG, E. Batch vs. online learning. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, Springer, p. 199–201, 2019. Citado na página 15.

CABRAL, G. G. et al. Class imbalance evolution and verification latency in just-in-time software defect prediction. In: IEEE. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. [S.l.], 2019. p. 666–676. Citado 2 vezes nas páginas 23 and 24.

CHAUDHARI, P. et al. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, IOP Publishing, v. 2019, n. 12, p. 124018, 2019. Citado na página 31.

CHEN, X. et al. Eile: Efficient incremental learning on the edge. In: IEEE. *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. [S.l.], 2021. p. 1–4. Citado na página 6.

DITZLER, G. et al. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, IEEE, v. 10, n. 4, p. 12–25, 2015. Citado na página 5.

DONG, F. et al. Concept drift region identification via competence-based discrepancy distribution estimation. In: IEEE. *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. [S.l.], 2017. p. 1–7. Citado na página 2.

ESHKOL-TARAVELLA, I. et al. Chunk different kind of spoken discourse: Challenges for machine learning. In: *Language Resources and Evaluation Conference*. [S.l.: s.n.], 2020. p. 5164–5168. Citado na página 11.

EYOLFSON, J.; TAN, L.; LAM, P. Do time of day and developer experience affect commit bugginess? In: *Proceedings of the 8th Working Conference on Mining Software Repositories*. [S.l.: s.n.], 2011. p. 153–162. Citado na página 20.

FAN, Y. et al. The impact of changes mislabeled by szz on just-in-time defect prediction.(2019). *IEEE Transactions on Software Engineering*, p. 1–26, 2016. Citado na página 21.

FERREIRA, P. M.; RUANO, A. E. Online sliding-window methods for process model adaptation. *IEEE Transactions on Instrumentation and Measurement*, IEEE, v. 58, n. 9, p. 3012–3020, 2009. Citado na página 9.

GAMA, J. et al. Learning with drift detection. In: SPRINGER. *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. [S.l.], 2004. p. 286–295. Citado na página 33.

GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine learning*, Springer, v. 90, p. 317–346, 2013. Citado 2 vezes nas páginas 7 and 39.

GARIPOV, T. et al. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, v. 31, 2018. Citado na página 32.

GOMES, H. M.; READ, J.; BIFET, A. Streaming random patches for evolving data stream classification. In: IEEE. *2019 IEEE international conference on data mining (ICDM)*. [S.l.], 2019. p. 240–249. Citado na página 11.

HAN, S. et al. Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security*, IEEE, v. 16, p. 2300–2311, 2021. Citado na página 10.

HARRIES, M.; WALES, N. S. et al. Splice-2 comparative evaluation: Electricity pricing. University of New South Wales, School of Computer Science and Engineering, 1999. Citado na página 37.

HINDER, F.; VAQUET, V.; HAMMER, B. One or two things we know about concept drift—a survey on monitoring in evolving environments. part a: detecting concept drift. *Frontiers in Artificial Intelligence*, Frontiers Media SA, v. 7, p. 1330257, 2024. Citado na página 7.

HOANG, T. et al. Deepjit: an end-to-end deep learning framework for just-in-time defect prediction. In: IEEE. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2019. p. 34–45. Citado 2 vezes nas páginas 23 and 25.

HOVAKIMYAN, G.; BRAVO, J. M. Evolving strategies in machine learning: A systematic review of concept drift detection. *Information*, MDPI, v. 15, n. 12, p. 786, 2024. Citado 2 vezes nas páginas 2 and 3.

IZMAILOV, P. et al. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018. Citado 4 vezes nas páginas 29, 31, 32, and 33.

JAIN, L. C. et al. A review of online learning in supervised neural networks. *Neural computing and applications*, Springer, v. 25, p. 491–509, 2014. Citado na página 10.

KADWE, Y.; SURYAWANSHI, V. A review on concept drift. *Iosr J. Comput. Eng*, v. 17, n. 1, p. 20–26, 2015. Citado na página 9.

KAMEI, Y. et al. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, Springer, v. 21, p. 2072–2106, 2016. Citado 2 vezes nas páginas 2 and 25.

KAMEI, Y.; SHIHAB, E. Defect prediction: Accomplishments and future challenges. In: IEEE. *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*. [S.l.], 2016. v. 5, p. 33–45. Citado 2 vezes nas páginas 20 and 22.

KAMEI, Y. et al. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, IEEE, v. 39, n. 6, p. 757–773, 2012. Citado 5 vezes nas páginas 2, 20, 22, 48, and 49.

KESKAR, N. S. et al. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. Citado na página 31.

KIM, S.; WHITEHEAD, E. J.; ZHANG, Y. Classifying software changes: Clean or buggy? *IEEE Transactions on software engineering*, IEEE, v. 34, n. 2, p. 181–196, 2008. Citado na página 48.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 29.

KOLTER, J. Z.; MALOOF, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, JMLR. org, v. 8, p. 2755–2790, 2007. Citado na página 12.

KRASNER, H. *The Cost of Poor Quality Software in the US: A 2022 Report*. [S.l.], 2022. Accessed: 2025-01-10. Disponível em: <https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report>. Citado na página 1.

KRAWCZYK, B.; CANO, A. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. *Applied Soft Computing*, Elsevier, v. 68, p. 677–692, 2018. Citado na página 8.

LANE, T.; BRODLEY, C. E. Approaches to online learning and concept drift for user identification in computer security. In: *KDD*. [S.l.: s.n.], 1998. p. 259–263. Citado na página 9.

LAURETIS, L. D. From monolithic architecture to microservices architecture. In: IEEE. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.], 2019. p. 93–96. Citado na página 23.

LEMOS, O. A. L. et al. Experience report: Can software testing education lead to more reliable code? In: IEEE. *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.], 2015. p. 359–369. Citado na página 1.

LESSMANN, S. et al. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE transactions on software engineering*, IEEE, v. 34, n. 4, p. 485–496, 2008. Citado na página 1.

LI, W. et al. Effort-aware semi-supervised just-in-time defect prediction. *Information and Software Technology*, Elsevier, v. 126, p. 106364, 2020. Citado na página 2.

LI, Z. et al. Incremental learning imbalanced data streams with concept drift: The dynamic updated ensemble algorithm. *Knowledge-Based Systems*, Elsevier, v. 195, p. 105694, 2020. Citado na página 6.

LOSING, V.; HAMMER, B.; WERSING, H. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, Elsevier, v. 275, p. 1261–1274, 2018. Citado na página 6.

LU, J. et al. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, IEEE, v. 31, n. 12, p. 2346–2363, 2018. Citado na página 2.

MCINTOSH, S.; KAMEI, Y. Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. In: *Proceedings of the 40th international conference on software engineering*. [S.l.: s.n.], 2018. p. 560–560. Citado 2 vezes nas páginas 2 and 23.

NOAA. *Fed. Climate Complex Global Surface Summary of Day Data–Version 7–Usaf Datsav3 Station N. 725540*. 2012. Accessed: Mar. 15, 2024. [Online]. Available: <https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ncdc:C00516>. Citado na página 38.

PACHOULY, J. et al. A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. *Engineering Applications of Artificial Intelligence*, v. 111, p. 104773, 2022. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0952197622000616>. Citado na página 1.

PALLI, A. S. et al. Online machine learning from non-stationary data streams in the presence of concept drift and class imbalance: A systematic review. *Journal of Information and Communication Technology*, v. 23, n. 1, p. 105–139, 2024. Citado na página 7.

PANDEY, A.; JADHAV, A. Towards effective software defect prediction using machine learning techniques. *SN Computer Science*, Springer, v. 5, n. 8, p. 1096, 2024. Citado na página 3.

PRATAMA, M.; ASHFAHANI, A.; HADY, A. Weakly supervised deep learning approach in streaming environments. In: *Proc. IEEE Int. Conf. Big Data*. Los Alamitos, CA, USA: IEEE Computer Society, 2019. p. 1195–1202. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/BigData47090.2019.9006285>. Citado 2 vezes nas páginas 10 and 15.

SCHLIMMER, J. C.; GRANGER, R. H. Incremental learning from noisy data. *Machine learning*, Springer, v. 1, p. 317–354, 1986. Citado na página 37.

SEBASTIÃO, R.; FERNANDES, J. M. Supporting the page-hinkley test with empirical mode decomposition for change detection. In: SPRINGER. *International Symposium on Methodologies for Intelligent Systems*. [S.l.], 2017. p. 492–498. Citado na página 9.

SHALEV-SHWARTZ, S. et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, Now Publishers, Inc., v. 4, n. 2, p. 107–194, 2012. Citado na página 9.

SHIHAB, E. et al. An industrial study on the risk of software changes. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. [S.l.: s.n.], 2012. p. 1–11. Citado na página 48.

ŚLIWERSKI, J.; ZIMMERMANN, T.; ZELLER, A. When do changes induce fixes? *ACM sigsoft software engineering notes*, ACM New York, NY, USA, v. 30, n. 4, p. 1–5, 2005. Citado 2 vezes nas páginas 20 and 21.

STREET, W. N.; KIM, Y. A streaming ensemble algorithm (sea) for large-scale classification. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2001. p. 377–382. Citado na página 37.

TABASSUM, S. et al. An investigation of cross-project learning in online just-in-time software defect prediction. In: *Proceedings of the acm/ieee 42nd international conference on software engineering*. [S.l.: s.n.], 2020. p. 554–565. Citado 2 vezes nas páginas 24 and 25.

TAN, M. et al. Online defect prediction for imbalanced data. In: IEEE. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. [S.l.], 2015. v. 2, p. 99–108. Citado 2 vezes nas páginas 24 and 25.

THOTA, M. K.; SHAJIN, F. H.; RAJESH, P. Survey on software defect prediction techniques. *International Journal of Applied Science and Engineering*, Chaoyang University of Technology, v. 17, p. 331–344, dezembro 2020. ISSN 1727-7841. Disponível em: <https://doi.org/10.6703/IJASE.202012_17(4).331>. Citado na página 1.

VINCENT, P. et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, v. 11, n. 110, p. 3371–3408, 2010. Citado na página 26.

WAHONO, R. S. A systematic literature review of software defect prediction. *Journal of software engineering*, v. 1, n. 1, p. 1–16, 2015. Citado na página 1.

WANG, B.; PINEAU, J. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 28, n. 12, p. 3353–3366, 2016. Citado na página 13.

ZHANG, H.-C.; WU, Q.; LI, F.-Y. Application of online multitask learning based on least squares support vector regression in the financial market. *Applied Soft Computing*, Elsevier, v. 121, p. 108754, 2022. Citado na página 10.

ZHANG, Z.-W.; JING, X.-Y.; WANG, T.-J. Label propagation based semi-supervised learning for software defect prediction. *Automated Software Engineering*, Springer, v. 24, p. 47–69, 2017. Citado na página 3.

ZHOU, Z.-H. et al. Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum]. *IEEE Computational intelligence magazine*, IEEE, v. 9, n. 4, p. 62–74, 2014. Citado na página 5.

ZHU, X. *Stream Data Mining Repository*. 2010. Accessed: Mar. 14, 2024. [Online]. Available: <http://www.cse.fau.edu/~xzhu/stream.html>. Citado na página 38.

ŽLIOBAITĖ, I. Combining similarity in time and space for training set formation under concept drift. *Intelligent Data Analysis*, IOS Press, v. 15, n. 4, p. 589–611, 2011. Citado na página 38.